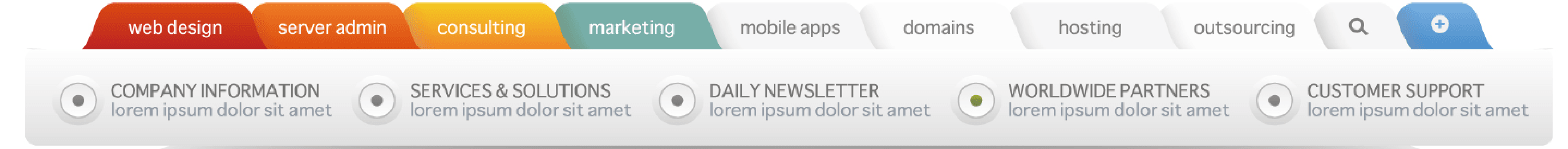
Web开发典藏大系



HTML 5+CSS 3 网页开发实战精解

12.6小时多媒体教学视频

杨习伟　等编著

手把手教会你用HTML 5和CSS 3开发网页

系统讲解HTML 5和CSS 3的新功能和新特性

理论结合实践，每个知识点都对应示例进行讲解

184个实例剖析、26个综合案例详解，提升实战技能



清华大学出版社

HTML 5+CSS 3 网页开发实战精解

杨习伟　等编著

清华大学出版社
北　京

内 容 简 介

本书全面、系统地讲解了 HTML 5 和 CSS 3 从 Web 界面设计到 Web 应用开发的各种技术。本书难度适中，学习梯度科学，知识架构严谨，内容由浅入深、从易到难，讲解通俗易懂，并注重读者兴趣的培养，讲解时还列举了大量实例，以帮助读者提高实战技能。本书配带 1 张光盘，内容为本书重点内容的教学视频和本书涉及的源代码。

本书共 17 章，分为 3 篇。第 1 篇为技术概览，简要介绍 HTML 5 标准和 CSS 3 层叠样式表等内容；第 2 篇为基于 CSS 3 的 Web 界面设计实战，重点介绍文字、背景、边框、盒布局、多列布局、动画、渐变、支持多种设备的样式表等内容；第 3 篇为基于 HTML 5 的 Web 应用开发实战，重点介绍绘图、音频和视频、新型表单、拖放、本地存储、离线应用、跨源通信、WebSocket 双向通信、多线程和地理位置等内容。

本书适合 Web 设计与开发的新手阅读，也适合有一定 Web 前端开发基础的网页开发人员阅读；对于大中专院校的学生，本书也不失为一本网页开发的好教材。如果阅读本书的读者具备 CSS 样式表和 JavaScript 的基础知识，学习效果会更好。

图书在版编目（CIP）数据

HTML 5+CSS 3 网页开发实战精解 / 杨习伟等编著. —北京：清华大学出版社，2013.1
（Web 开发典藏大系）
ISBN 978-7-302-28867-1

Ⅰ. ①H…　Ⅱ. ①杨…　Ⅲ. ①超文本标记语言–程序设计②网页制作工具　Ⅳ. ①TP312②TP393.092

中国版本图书馆 CIP 数据核字（2012）第 104736 号

责任编辑：夏兆彦
封面设计：欧振旭
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张：**24.75　**字　　数：**612 千字
（附光盘 1 张）
版　　次：2013 年 1 月第 1 版　　**印　　次：**2013 年 1 月第 1 次印刷
印　　数：1～ 000
定　　价：　　元

产品编号：047124-01

前　　言

HTML 5 和 CSS 3 代表了下一代的 HTML 和 CSS 技术，它们必将推动互联网的快速发展。无论是移动开发，还是云计算，HTML 5 担负着不可替代的使命，已经延伸到各个应用领域。对于界面而言，CSS 3 则是首选技术。

目前，当 HTML 5 和 CSS 3 的相关规范还在不断完善时，各大浏览器厂商就已经紧锣密鼓地完善浏览器的功能，以便更好地支持最新的规范。由此可以看出，各大厂商对这二者是多么重视。也许有人觉得，HTML 5 和 CSS 3 尚未成熟，并且很多用户仍然在使用早期的浏览器，所以不适合进行大规模开发和应用。但其实在 HTML 5 的规范中，已经考虑了过渡性的问题，它不但有超强的兼容性，还以一种引导的方式引导用户升级浏览器，可以预见，在不久的将来，它必将开创一个新的互联网时代，并将成为主流的技术之一。

沿着这个发展趋势，不久的将来，Web 前端开发的含金量会越来越高，门槛也会越来越高，Web 前端技术必将进入一个崭新的时代。而正是 HTML 5 和 CSS 3 开启了这扇大门。对于任何一个想从事或者正在从事 Web 前端开发的人员而言，HTML 5 和 CSS 3 已经是必须掌握的前沿技术了。基于此，作者总结了已经公布的 HTML 5 和 CSS 3 的相关技术规范和标准，并结合自己多年的 Web 前端开发经验，写成了本书，希望为想学习 HTML 5 和 CSS 3 技术的读者提供必要的帮助。另外，HTML 5 和 CSS 3 仍然在发展之中，很多人对此知之甚少，甚至有很多误解或偏见，希望通过本书以正视听，帮助有志之士和好学之人进入这个崭新的领域。

本书特色

1. 提供配套的多媒体教学视频

本书中的重点内容都专门录制了配套的多媒体视频，以帮助读者更加直观而高效地学习，从而达到事半功倍的效果。

2. 内容架构独特、科学，构思巧妙、新颖

本书每章都集中探讨一组可以通过 HTML 5 和 CSS 3 解决的特定问题，而区别于传统图书中将 HTML 5 和 CSS 3 技术机械罗列。通过这种主题归纳的方法，可以让读者感受实际的网页开发是如何进行的，从而提高实战水平。

3. 内容新颖，纳入了最新的技术规范

本书以当前 Web 开发中最热门的 HTML 5 和 CSS 3 为写作版本，写作时查阅了大量官方英文资料中的 HTML 5 和 CSS 3 最新发布的规范和标准，并纳入本书内容，让本书得以展示最新技术的魅力。

4．内容全面、系统

本书内容涉及 HTML 5 和 CSS 3 从 Web 界面设计到 Web 应用开发的各种技术，涵盖了 Web 前端设计需要的方方面面技术。掌握这些技术，便可以胜任相关的开发工作。

5．难度适中，讲解由浅入深，学习梯度科学

本书内容遵循技术概览→Web 界面开发→Web 应用开发的学习梯度，讲解由浅入深，由易到难，逐层推进，学习梯度平滑，读者容易掌握。阅读本书，读者会有一种轻轻松松的感觉。

6．以实例讲解，用插图说话，提高实战技能

本书各章的重点内容都穿插了大量实例进行讲解，引导读者学习，帮助读者更好地理解相关技术要点，快速掌握实际开发中的各种实战技能。另外，本书在讲解时，尽可能提供丰富的插图，这样可以更加直观地理解知识，学习起来也更加快捷。

7．娓娓道来，轻松愉快，趣味性强

本书采用幽默和风趣的讲述方式来讲解，娓娓道来，阅读起来轻松愉快。书中的每句话都力求通俗易懂，避免那种云山雾罩式的讲解。这种活泼的风格，可以激发读者的学习兴趣，达到更好的学习效果。

内容概览

第1篇　技术概览（第1章、第2章）

本篇简单介绍了 HTML 5 和 CSS 3 的发展背景及发展现状，并介绍了相关的基础知识，其中重点讲解了选择器的功能。

第2篇　基于CSS 3的Web界面设计实战（第3~7章）

本篇主要介绍了基于 CSS 3 的 Web 页面设计，涉及文字、背景、边框、盒布局、多列布局、动画、渐变、支持多种设备的样式表等内容，尤其重点介绍了 CSS 3 在 Web 页面设计方面新增的很多有用的功能。学完本篇，便可以设计非常精美的网页了，如阴影、圆角等设计将变得非常简单。

第3篇　基于HTML 5的Web应用开发实战（第8~17章）

本篇主要介绍基于 HTML 5 的 Web 应用开发，尤其重点介绍 HTML 5 新增的 Web 应用功能，主要包括绘图、多媒体、表单、拖放、消息传递、本地存储、离线应用、WebSocket 通信、多线程、地理位置等。本篇的每章都是一个完整的模块，掌握了这些内容，可以开发比较复杂的 Web 应用。

光盘内容

- 本书重点内容的教学视频；

- 本书实例涉及的源代码。

对读者的建议

- 学习 Web 前端开发，首先要从整体上了解 Web 前端涉及的东西，然后逐一掌握各种技术，一一击破。
- 本书提供了独特的内容架构，建议新手从头开始顺次逐章阅读，尽量不要跳跃。有基础的人可以根据自己的需要有选择性地阅读。
- 本书每章讲解的主题都结合了实例，对于这些例子，一定要亲自动手做一做，才能理解得更深刻。
- 对于重点、难点内容，建议读者结合本书的配套教学视频阅读，效果更佳。

本书读者对象

- Web 前端开发人员；
- Web 设计与开发爱好者；
- 网页设计与开发人员；
- 各大中专院校的学生；
- 培训班的学员。

本书作者

本书由杨习伟主笔编写。其他参与编写的人员有毕梦飞、蔡成立、陈涛、陈晓莉、陈燕、崔栋栋、冯国良、高岱明、黄成、黄会、纪奎秀、江莹、靳华、李凌、李胜君、李雅娟、刘大林、刘惠萍、刘水珍。在此表示感谢！

阅读本书的过程中若有疑问，请发邮件和我们联系。E-mail：bookservice2008@163.com。

目录

第 1 篇　技术概览

第 2 篇 基于 CSS 3 的 Web 界面设计实战

第 3 篇 基于 HTML 5 的 Web 应用开发实战

第 1 篇　技术概览

第 1 章　HTML 5 标准

HTML 5 是下一代 HTML 的标准，目前仍然处于发展阶段。经过了 Web 2.0 时代，基于互联网的应用已经越来越丰富，同时也对互联网应用提出了更高的要求。今天，技术人员、设计者、互联网爱好者都在热议 HTML 5，HTML 5 俨然已经成为互联网领域最热门的词语。与 HTML 4 相比，HTML 5 的发展有着革命性的进步。基于良好的设计理念，HTML 5 不但增加了很多新功能，而且对于涉及的每一个细节都有着明确的规定。HTML 5 正在引领时代的潮流，必将开创互联网的新时代。

1.1　HTML 5 介绍

在学习 HTML 5 之前，我们先介绍一下 HTML 5 的来龙去脉。

1.1.1　HTML 5 的历史背景

HTML 的出现由来已久。1993 年，HTML 首次由因特网工程任务组（IETF）以因特网草案的形式发布。接着，HTML 的发展一路高歌：1995 年发布了 2.0 版，1996 年发布了 3.2 版，1997 年发布了 4.0 版，1999 年 12 月发布了 4.01 版。从第三个版本（3.2 版）开始，W3C（万维网联盟）开始接手，并负责后续版本的制定工作。

在 HTML 4.01 之后，W3C 的认识发生了倒退，把发展 HTML 放在了次要地位，而把主要注意力转移到了 XML 和 XHTML 之上。由于当时正值 CSS 崛起，设计者们对于 XHTML 的发展深信不疑。但随着互联网的发展，HTML 迫切需要增加一些新的功能，制定新的规范。

为了能继续深入发展 HTML 规范，在 2004 年，一些浏览器厂商联合成立了 WHATWG（Web 超文本技术工作小组），以推动 HTML 5 规范。最初，WHATWG 的工作内容包含两个部分：Web Forms 2.0 和 Web Apps 1.0。它们都是对 HTML 的发展，并被纳入 HTML 5 的规范之中。Web 2.0 也是在那个时候提出来的。

2006 年，W3C 组建了新 HTML 的工作组，非常明智地采纳了 WHATWG 的意见，于 2008 年发布了 HTML 5 的工作草案。2009 年 W3C 停止了对 XHTML 2 的工作。紧接着，HTML 5 的草案不断地更新。2010 年，HTML 5 开始进入众多开发者的视野。在 HTML 5 规范还没定稿的情况下，各大浏览器厂商就已经按捺不住了，纷纷参与到规范的制定中来，并对自己旗下的产品进行升级以支持 HTML 5 的新功能。

HTML 5 规范涉及的内容非常多，众多浏览器厂商的参与使得可以及时获得一些实验性的反馈，HTML 5 规范也得以持续的完善。与此同时，IETF 制定相关通信协议。HTML

5 就是以这种方式快速地融入到 Web 开发平台当中。

1.1.2　HTML 5 的现状

今天，几乎所有最新版的浏览器都已经在不同程度上支持 HTML 5，开发者也可以尝试开发基于 HTML 5 的新功能了。

1. 两个发展的时间点

HTML 5 规范仍然处于草案更新之中，但是最新的草案已经相对比较完备了。HTML 规范的发布有两个时间点，如图 1-1 所示。



图 1-1　HTML 5 规范的发布时间点

第一个时间点是 2012 年，目标是发布“候选推荐版”；第二个时间点是 2022 年，目标是发布“计划推荐版”。

也许你会认为，既然要到 2022 年，那就等十年再说吧。这是非常错误的想法，你需要先明白这两个时间点的真正含义再下结论。

首先是 2012 年，这是最重要的时间点，因为这一年要发布候选推荐版。一个规范如果要成为候选推荐标准（REC），它需要具备百分之百的可实施性，只有成功通过上万项的测试案例后才能验证这一点。据保守估计，整个规范可能需要进行 2 万项测试。

其次是 2022 年，这一年要发布计划推荐版。那个时候，至少会有两大浏览器会完全支持 HTML 5 的所有特性。如果考虑到 HTML 5 标准的规模，这个日期还是太乐观了。毕竟之前发布的 HTML 4.01 已经经过十多年都没能实现这一目标。

按照此说法，在两大浏览器支持所有的功能之前，HTML 5 的规范是没有最终定稿的。如果你真的等十年之后再来学习，就悔之晚矣。

2. 目前发展状况

首先，各大浏览器都已经积极支持 HTML 5 的新功能。各大浏览器厂商不仅积极地支持 HTML 5 规范，而且还参与到 HTML 5 规范的制定之中。真正重要的一小部分 HTML 5 新特性已经得到浏览器的支持。所有浏览器对 HTML 5 的支持情况，都会在一周之内发生变化，因为浏览器制作厂商的创新速度也非常快。

其次，目前已经存在很多基于 HTML 5 的应用了。有些网站明确表示支持 HTML 5，并提示用户升级浏览器，很多网络应用也已经为 HTML 5 的到来做了充分的准备。2012 年是 HTML 5 真正发力的开始，而且对于移动应用优势非常明显。

3. HTML 5的开发组织

HTML 5 最初是由 WHATWG 推动开发的，到了今天，已经有三个重要的组织参与进来。

- WHATWG：由来自 Apple、Mozilla、Opera、Google 等浏览器厂商的人员组成，

成立于 2004 年。负责开发 HTML 和 Web 应用 API，并提供了开放式的合作。

- W3C：W3C 下辖的 HTML 工作组负责发布 HTML 5 规范。
- IETF：因特网工程任务组下辖的 HTTP 等负责 Internet 协议的团队。其中 HTML 5 定义的一个新功能 WebSocket API 就依赖于新的 WebSocket 协议，这个协议正是由 IETF 开发的。

1.1.3　良好的设计理念

回顾 HTML 的发展历程，曾经出现了 XHTML 规范，但没有得到较好的发展。特别是 XHTML 2，对语法解析过于严格，严重地违背了著名的 Postel 法则：“发送时要保守；接收时要开放”。

根据 Postel 法则，对自己发送出去的东西要严格要求，而对接收的东西则要放松限制。HTML 5 的设计理念就遵循了这个法则，同时 HTML 5 也强调了其兼容性、实用性和互操作性。

1. 化繁为简

HTML 5 为了做到尽可能简化，避免了一些不必要的复杂设计。例如，DOCTYPE 的改进：在过去的 HTML 版本中，第一行的 DOCTYPE 过于冗长，没有几个人能记得住，在实际的 Web 开发中也没有什么意义。而在 HTML 5 中就非常简单：

```
<!DOCTYPE html>
```

HTML 5 改进的方面如下。

- 简化了 DOCTYPE。
- 简化了字符集声明。
- 以浏览器的原生能力替代脚本代码的实现。
- 简单而强大的 HTML 5 API。

为了让一切变得简单，HTML 5 可谓下足了功夫。为了避免造成误解，HTML 5 对每一个细节都有着非常明确的规范说明，不允许有任何的歧义和模糊出现。

2. 向下兼容

遵循 Postel 法则，HTML 5 有着很强的兼容能力。在这方面，HTML 5 没有颠覆性的革新，允许存在不严谨的写法，例如，一些标签的属性值没有使用引号括起来；标签属性中包含大写字母；有的标签没有闭合等。然而这些不严谨的错误处理方案，在 HTML 5 的规范中都有着明确的规定，也希望未来在浏览器中有一致的支持。当然对于 Web 开发者来说，还是严谨一点好。

对于 HTML 5 的一些新特性，如果旧的浏览器不支持，也不会影响页面的显示。在 HTML 5 的规范中，也考虑了这方面的内容。如在 HTML 5 中 input 标签的 type 属性增加了很多类型，当浏览器不支持这些类型时，默认会将其视为 text，实现了优雅的降级。

3. 支持合理存在的内容

HTML 5 的设计者们花费了大量的精力来研究通用的行为。例如，Google 分析了上百

万的页面，从中提取了 DIV 标签的 ID 名称。例如很多人都是这样标记导航区域的：

```
<div id="nav">
    //导航区域内容
</div>
```

既然该行为已经大量存在，HTML 5 就会想办法去改进，所以就直接增加了一个 nav 标签，用于导航区域。

4. 解决实用性的问题

对于 HTML 无法实现的一些功能，用户会寻求其他方法来实现，如对于绘图、多媒体、地理位置、实时获取信息等应用，通常会开发一些相应的插件间接地去实现。HTML 5 的设计者们研究了这些需求，开发了一系列用于 Web 应用的接口。

HTML 5 规范的制定是非常开放的，所有人都可以获取草案的内容，也可以参与进来提出宝贵的意见。因为开放，所以可以得到更加全面的发展。一切以用户需求为最终目的，用户需要什么，它就规范什么。所以，当你在使用 HTML 5 的新功能时，会发现正是期待已久的东西。

5. 最终用户优先

这个设计理念本质上是一种解决冲突的机制。在遇到无法解决的冲突的时候，规范会把最终用户的诉求放在第一位。因此，HTML 5 的绝大部分功能都是非常实用的。HTML 5 规范的制定遵循如下的优先顺序：

```
用户 > 编写 HTML 的开发者 > 浏览器厂商 > 规范制定者 > 理论的存粹性
```

可见，用户与开发者的重要性要远远高于规范和理论。例如，有很多用户都需要实现一个新的功能，HTML 5 规范设计者们会研究这种需求，并纳入规范；HTML 5 规范了一套错误处理机制，以便当 Web 开发者写了不够严谨的代码时，接纳这种不严谨的做法。

所以，当你在使用 HTML 5 时，会发现它比以前的版本友好多了。

6. 通用访问

这个原则可以分成如下三个方面。

- 可访问性：HTML 5 考虑了残障用户的需求，以屏幕阅读器为基础的元素也已经被添加到规范当中。
- 媒体中立：HTML 5 规范不仅仅是为某些浏览器而设计的。也许有一天，HTML 5 的新功能在不同的设备和平台上都能运行。
- 支持所有语种：例如，新的<ruby>元素支持在东亚页面的排版中会用到的 Ruby 注释。

1.1.4　新增的 HTML 5 原生功能

1. 新增的原生功能

Web 应用程序是 HTML 5 中最大的亮点，它原生地支持了一些只有在桌面应用中才能

实现的功能。这些功能主要如下。

- Canvas 绘图（2D 和 3D）。
- Channel 消息传送。
- 跨文档消息传送。
- 地理位置。
- WebSocket API 及其协议。
- 本地存储。
- 本地数据库。
- Web 工作线程。
- 跨域的异步请求。
- 离线应用。

这些功能都有着独立的规范，并且从属于 HTML 5。

2. 原生功能的优点

在过去，很多功能只能通过安装插件才能够实现。插件的方式会存在很多问题。

- 插件需要安装，且安装过程可能失败。
- 插件可能会被屏蔽或禁用。
- 插件本身存在安全隐患。
- 插件不容易与 HTML 文档的其他部分集成。
- 插件的开发，对开发者的技能要求进一步提高。

而 HTML 5 的原生功能，让这些功能的实现变得简单，不需要安装不安全的插件，而且能与页面中的其他元素无缝集成，安全性也提高了很多。

1.1.5　HTML 5 带来的好处

对于用户和开发者而言，HTML 5 的出现意义非常重大。因为它将解决之前 Web 页面存在的诸多问题。

首先，不必考虑各个浏览器的兼容性问题。

在 HTML 5 之前，各大浏览器厂商为了争夺市场占有率，会在各自的浏览器中增加各种各样的功能，并且不具有统一的规范。对于用户而言，使用不同的浏览器，常常会看到不同的页面效果，甚至有些功能根本不能使用；Web 开发者也伤透脑筋，为了使页面能在多个浏览器中正常使用，不得不写一些 hack，徒增了 Web 开发的复杂度。

在 HTML 5 中，纳入了所有合理的扩展功能。HTML 5 规范的内容也非常庞大，各大浏览器厂商关注的是自己的产品能否更好地支持 HTML 5。这样，只要是基于 HTML 5 开发的 Web 应用，都能够在浏览器中正常显示（不过，现在还不能完全这样做）。

其次，可以实现复杂的 Web 应用。

在 HTML 5 之前，HTML 与 Web 应用程序的关系十分薄弱。与强大的桌面应用程序的功能相比，Web 应用程序的功能是微不足道的，并且在很多方面都做了限制。

在 HTML 5 中，不但大大扩展了 Web 应用的功能，而且安全性也有了明确的规范。各大浏览器也争相封装了这些功能，目前已经可以使用 HTML 5 开发富 Web 应用了。

1.2　全新的 HTML 5

下面让我们快速预览一下 HTML 5 的新功能。

1.2.1　从“头”说起

这里所说的是 HTML 文件的开头部分。HTML 5 避免了不必要的复杂性，DOCTYPE 和字符集都极大地简化了。

1. 简化的DOCTYPE声明

DOCTYPE 声明是 HTML 文件中必不可少的内容，它位于文件的第一行，声明了 HTML 文件遵循的规范。声明 HTML 4.01 版的 DOCTYPE 代码为：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

这么长的一串代码恐怕极少有人能够默写出来，通常都是通过复制/粘贴的方式添加这段代码。而在 HTML 5 中的 DOCTYPE 代码则非常简单：

```
<!DOCTYPE html>
```

现在好记得多了，可以告别复制/粘贴的时代了。同时这种声明，也标志性地让人感觉到这是符合 HTML 5 规范的页面。如果使用了 HTML 5 的 DOCTYPE 声明，则会触发浏览器以标准兼容的模式来显示页面。

2. 简化的字符集声明

字符集的声明也是非常重要的，它决定了页面文件的编码方式。在过去，都是使用如下的方式来指定字符集的：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

HTML 5 把它简化成了：

```
<meta charset="utf-8" />
```

在 HTML 5 中，以上两种方式都可以使用，这是由 HTML 5 的向下兼容的原则决定的。

1.2.2　明确简洁的结构

一个非常典型的页面设计中通常会包含头部、页脚、导航、主体内容和侧边内容等区域。Google 从上百万的网页中分析了其存在的合理性，例如，头部区域通常使用：

```
<div id="header">
    //页眉
</div>
```

针对这种情况，HTML 5 引入了与文档结构相关联的结构元素，如表 1.1 所示。

表 1.1　HTML 5 中的结构元素

元 素 名 称	描 述 说 明
header	标记头部区域的内容
footer	标记页脚区域的内容
section	Web 页面中的一块区域
article	独立的文章内容区域
aside	相关侧边内容或者引文区域
nav	导航类内容区域

下面我们就使用这几个新的结构元素来构建一个新型的博客页面，并且使用前面介绍的 DOCTYPE 和字符集。

【示例 1-1】 HTML 5 页面结构示例。

```
<!DOCTYPE HTML>
<html>
<head>
<title>HTML 5 示例</title>
<meta charset="utf-8">
<link rel="stylesheet" href="Code1-1.css" />
</head>
<body>
<header>
  <h1>清茶博客</h1>
  <p>喝一杯清茶，约三五知己，聊聊天，叙叙旧...</p>
</header>
<nav>
  <ul>
    <li><a href="#">首页</a></li>
    <li><a href="#">博客</a></li>
    <li><a href="#">相册</a></li>
    <li><a href="#">个人档案</a></li>
  </ul>
</nav>
<div id="container">
  <section>
    <article>
      <header>
        <h1>HTML 5</h1>
      </header>
      <p>HTML 5 是下一代 HTML 的标准，目前仍然处于发展阶段。经过了 Web2.0 时代，基于
互联网的应用已经越来越丰富，同时也对互联网应用提出了更高的要求。</p>
      <footer>
        <p>编辑于 2012.1.18</p>
      </footer>
    </article>
    <article>
      <header>
        <h1>CSS3</h1>
      </header>
      <p>对于前端设计师来说，虽然 CSS3 不全是新的技术，但它却重启了一扇奇思妙想的窗口。
      </p>
      <footer>
```

```
        <p>编辑于 2012.1.19</p>
      </footer>
    </article>
  </section>
  <aside>
    <article>
      <h1>简介</h1>
      <p>HTML 5 和 CSS3 正在掀起一场变革，它不是在替代 Flash， 而是正在发展成为开放的
Web 平台，不但在移动领域建功卓著，而且对传统的应用程序发起挑战。</p>
    </article>
  </aside>
  <footer>
    <p>版权所有 2012</p>
  </footer>
</div>
</body>
</html>
```

如示例 1-1 所示的代码，与 HTML 5 之前的页面布满 div 标签相比，已经变得清晰了很多。涉及的各种头部，我们都会将其包含在 header 标签内，各种结尾内容都会使用 footer 标签包含，导航菜单放在 nav 标签内，主要内容放在 section 标签内，独立的文章部分放在 article 标签内，相关的简介等内容放在侧边标签 aside 内。

示例 1-1 中引用了一个 CSS 文件（Code1-1.css），这里我们不再展示 CSS 文件内容（我们会提供相应的源代码供读者学习之用）。运行结果如图 1-2 所示。



图 1-2　新型的 HTML 5 页面结构

使用这些用于结构布局的元素，在设计样式表的时候，不用再添加标签的 id 特性作为

选择器了，而是直接针对标签进行设计，因为标签已经具有一定的语义。

1.2.3　新增的元素

上一节介绍了 HTML 5 中的一些结构化的元素，使得页面布局耳目一新。本节将全面介绍 HTML 5 中新增的元素。HTML 5 新增了很多新的有意义的标签，为了方便记忆，我们对它们进行了分类。

1. 结构片段

- article：标识独立的主体内容区域，可用于论坛帖子、报纸文章、博客条目、用户评论等。
- aside：标识非主体内容区域，该区域中的内容应该与附近的主体内容相关。
- section：标识文档的小节或部分。
- footer：标识页面的页脚，或内容区块的脚注。
- header：标识页面的页首，或内容区块的标头。
- nav：标识页面的导航区块。

2. 进度信息

- meter：根据 value 属性赋值和最大、最小值的度量进行显示的进度条状条形图。
- progress：标识任务进度显示的进度条。

3. 交互性元素

- command：标识一个命令元素（单选、复选或者按钮）；当且仅当这个元素出现在<menu>元素里面时才会被显示，否则将只能作为键盘快捷方式的一个载体。
- datalist：标识一个选项组，与 input 元素配合使用该元素，来定义 input 可能的值。

4. 内嵌应用元素及辅助元素

- audio：定义声音，如音乐或其他音频流。
- video：定义视频，如电影片段或其他视频流。
- source：为媒介元素（如 video 和 audio）定义媒介资源。
- track：为诸如 video 元素之类的媒介规定外部文本轨道。
- canvas：定义图形，比如图表和其他图像。该标签只是图形容器，必须使用脚本来绘制图形。
- embed：标识来自外部的互动内容或插件。

5. 在文档和应用中使用的元素

- details：标识描述文档或文档某个部分的细节。
- summary：标识<details>元素的标题。
- figcaption：标识 figure 元素的标题。
- figure：标识一块独立的流内容（图像、图表、照片、代码等）。

- hgroup：标识文档或内容的多个标题。用于将 h1～h6 元素打包，优化页面结构在 SEO 中的表现。

6. ruby标签

- ruby：标识 ruby 注释（中文注音或字符）。在东亚使用，显示的是东亚字符的发音。
- rp：在 ruby 注释中使用，以定义不支持 ruby 元素的浏览器所显示的内容。
- rt：标识字符（中文注音或字符）的解释或发音。

7. 文本和文本标记元素

- bdi：允许设置一段文本，使其脱离其父元素的文本方向设置。
- mark：标识需高亮显示的文本。
- time：标识日期或时间。
- output：标识一个输出的结果。

8. 其他

- keygen：标识表单密钥生成器元素。当提交表单时，私钥存储在本地，公钥发送到服务器。
- wbr：标识单词中适当的换行位置；可以用此标签为一个长单词指定合适的换行位置。

1.2.4　废弃的元素

HTML 5 也删除了一些元素，主要是以下几个方面的元素。

1. 能使用CSS方案替代的元素

在 HTML 5 之前的一些元素中，有一部分是纯粹用作显示效果的元素。而 HTML 5 延续了内容与表现的分离，对于显示效果更多地交给 CSS 去完成。所以，在这方面废除的元素有：basefont、big、center、font、s、strike、tt、u。

2. 不再支持frame框架

由于 frame 框架对网页可用性存在负面影响，因此在 HTML 5 中已经不支持 frame 框架，但支持 iframe 框架。所以 HTML 5 删除了 frame 框架中的 frameset、frame、noframes 元素。

3. 其他被废除的元素

其他元素被废除的原因通常都是有了较好的替代方案。

- 废除的 applet 元素：可由 embed 和 object 元素替代。
- 废除的 bgsound 元素：可由 audio 替代。
- 废除的 marquee 元素：可由 JavaScript 编程方式替代。
- 废除的 rb 元素：可由 ruby 元素替代。

- 废除的 acronym 元素：可由 abbr 元素替代。
- 废除的 dir 元素：可由 ul 元素替代。
- 废除的 isindex 元素：可以使用 form 元素和 input 元素相结合的方式替代。
- 废除的 listing 元素：可由 pre 元素替代。
- 废除的 xmp 元素：可由 code 元素替代。
- 废除的 nextid 元素：可由 GUIDS 替代。
- 废除的 plaintext 元素：可由“text/plain”MIME 类型替代。

在 HTML 5 继续完善的过程中，可能还会废除其他不合理的元素。

1.2.5　全新的选择器

HTML 5 极大地增强了选择器的功能。在 HTML 5 之前，如果要在页面中查找特定元素，只能使用三个函数：getElementById()、getElementsByName()、getElementsByTagName()。

1. 根据类名匹配元素（DOM API）

HTML 5 新增了 getElementsByClassName()函数，是根据类目匹配元素的，返回的是匹配到的数组，无匹配则返回空的数组。

```
var els = document.getElementsByClassName('section');
```

支持浏览器：IE9、Firefox 3.0+、Safari 3.2+、Chrome 4.0+、Opera 10.1+。

2. 根据CSS选择器匹配元素（Selectors API）

HTML 5 还提供了两个根据 CSS 选择器匹配元素的函数：querySelector()和 query SelectorAll()。

querySelector()返回匹配到的第一个元素，如果没有匹配则返回 null。

```
var els = document.querySelector("ul li:nth-child(odd)");
var els = document.querySelector("table.test > tr > td");
var els = document.querySelector(".class1",”.class2”);
```

querySelectorAll()返回所有匹配到的元素数组，如果没有匹配则返回空的数组。

```
var els = document.querySelectorAll("ul li:nth-child(odd)");
var els = document.querySelectorAll("table.test > tr > td");
var els = document. querySelectorAll (".class1",".class2");
```

querySelector()和 querySelectorAll()函数的参数可以接受两个或两个以上，只要满足任何一个条件都是有效的。支持浏览器：IE8+、Firefox 3.5+、Safari 3.2+、Chrome 4.0+、Opera 10.1+。

1.2.6　脚本日志和调试

JavaScript 脚本的调试一直都是这个开发语言的薄弱之处。基于 HTML 5 的应用开发，会大量使用 JavaScript 脚本，所以如何调试这些代码变得非常重要。所幸脚本日志功能会

给开发人员带来很多便利。

1. 脚本日志

脚本日志使用的是 console.log()函数，把信息输出到控制台。使用方法如下：

```
console.log("Hello World!");
console.log(document);       //输出 document 对象
```

可以输出字符串，也可以输出对象。如果输出的是对象，在控制台可以查看该对象的详细信息。与使用 alert()输出消息相比，console.log()的控制台输出不会阻塞脚本的执行。

脚本日志功能已经成为 JavaScript 开发人员常用的调试方法。为了便于开发人员查看输出到控制台的信息，很多浏览器都提供了控制台查看功能，如图 1-3 所示。

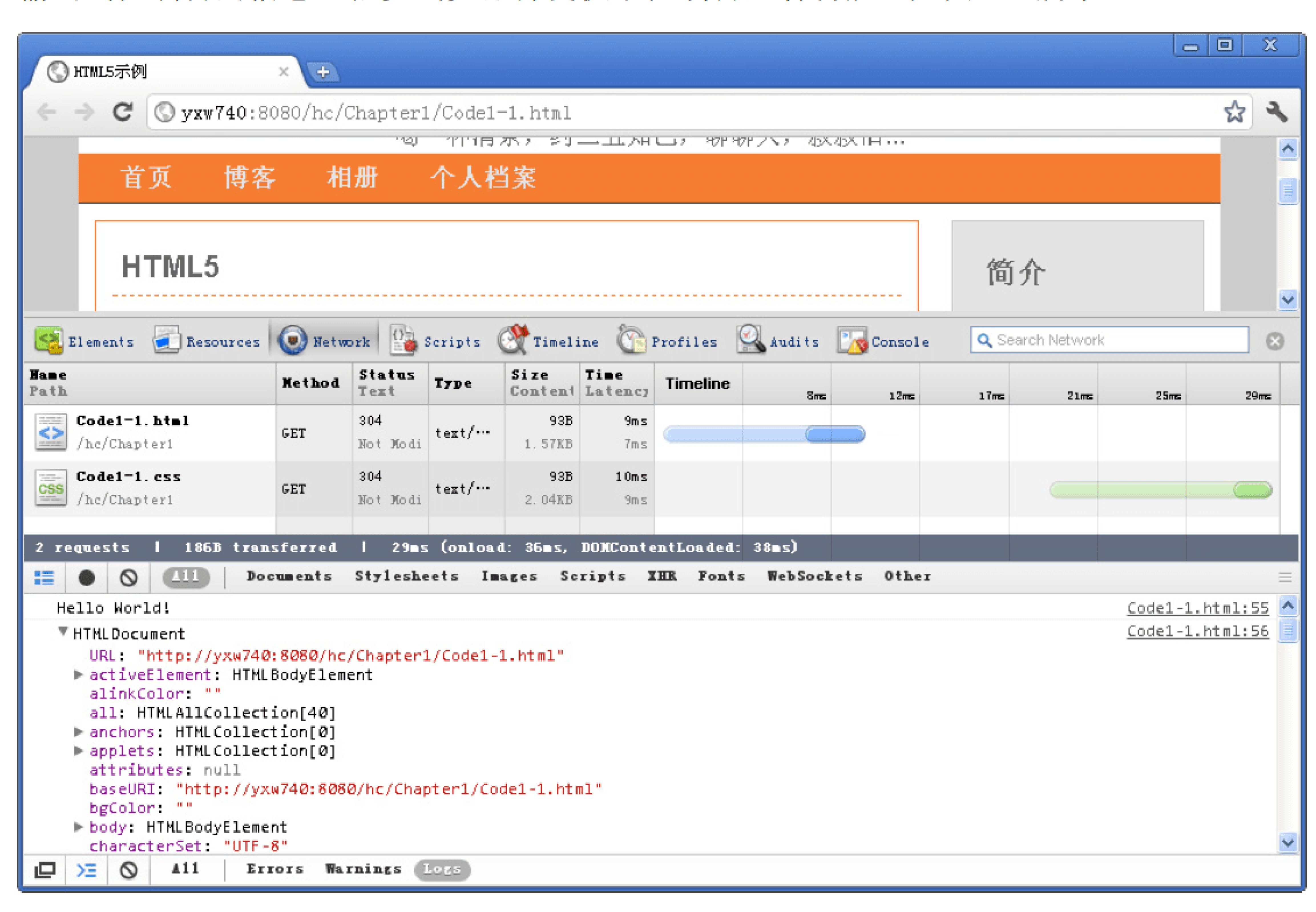


图 1-3　Chrome 浏览器中的开发者工具

2. 脚本调试

说起脚本调试，很多开发人员会首先想到 Firefox 浏览器中的一款 Firebug 插件。现在很多浏览器中，也内嵌了具有相同功能的开发工具：Safari 的 Web Inspector、Google 的 Chrome 开发者工具（Developer Tools）、IE 的开发者工具（Developer Tools），以及 Opera 的 Dragonfly 等。图 1-3 所示的是 Chrome 浏览器中的开发者工具。

这些开发者工具不但能够查看控制台，而且能够查看资源视图、存储视图、脚本调试视图、时间视图等，并且很多开发者工具都已经支持断点调试的功能。这些功能极大地提高了开发人员的开发效率。

1.3　HTML 5 的未来发展

HTML 5 的未来发展是极具野心的，绝非常规的富客户端应用那么简单。

1. 什么是RIA

RIA（Rich Internet Applications）即富互联网应用，具有高度互动性、丰富用户体验及功能强大的客户端。RIA 最突出的特点就是 Rich，它包含了两层含义：丰富的数据模型和丰富的界面元素。

RIA 技术提供了多种数据模型来处理客户端复杂的数据操作。使用 RIA 可以将部分原本需要在后台程序处理的问题转移到客户端，使数据能够被缓存在客户端，从而可以实现一个比基于 HTML 的响应速度更快，且数据往返于服务器的次数更少的用户界面。

RIA 技术也提供了比 HTML 更为丰富的界面表现元素，密集、响应速度快和图形丰富的页面元素与数据模型结合在一起，为用户提供好的使用体验。

RIA 具有桌面应用程序的特点：在消息确认和格式编排方面提供互动用户界面；在无刷新页面之下提供快捷的界面响应时间；提供通用的用户界面特性如拖放式（drag and drop），以及在线和离线操作能力。

RIA 具有 Web 应用程序的特点：立即部署、跨平台、采用逐步下载方式来检索内容和数据，以及可以充分利用被广泛采纳的互联网标准。RIA 具有通信的特点，包括实时互动的声音和图像。

客户端在RIA中的作用不仅仅是展示页面，还可以在幕后与用户请求异步地进行计算、传送和检索数据、显示集成的用户界面和综合使用声音和图像，这一切都可以在不依靠客户机连接的服务器或后端的情况下进行。

毫无疑问，HTML 5 汲取了 RIA 中的技术特点，并统一在浏览器中实现。

2. 与Flash和Silverlight之争

在 HTML 5 之前，出现了大批的 RIA 客户端开发技术，这也直接威胁着 HTML 在互联网中的地位。但是这些 RIA 技术来自于不同的开发商，没有统一的规范。其中比较有实力的 RIA 客户端开发技术就有近十种。

- Adobe Flash/Flex
- Silverlight
- Laszlo
- Avalon
- Java SWT
- XUL
- Bindow
- JavaFX
- Curl

这其中的 Flash 和 Silverlight 算是比较主流的技术了，均有着庞大的开发者群体。

HTML 5 发展了一大批原生的功能，完全可以替代 Flash 和 Silverlight 的实现。由于这些 RIA 技术的应用都是以插件的形式依附着浏览器运行的，因此 HTML 5 的出现对其造成

了极大的威胁。其中，Adobe 已经宣布放弃 Flash 移动业务，最大的原因是因为有了 HTML 5。

事实上，HTML 5 的功能在很多方面并不成熟，再加上 Flash 和 Silverlight 都有着庞大的用户群体，HTML 5 和 Flash、Silverlight 等 RIA 技术仍然会长期存在，只不过 HTML 5 顺应了发展的趋势，它的绝对优势在于未来。

3. 最终方向

在技术发展方面，HTML 5 将会把桌面应用所能实现的功能逐渐地移植进来：包括 3D 绘图、设备元素、触摸时间、点对点通信等。

随着网络带宽的提高和各种移动设备的流行，将会出现越来越多的 Web 应用程序。Web 应用程序的好处就是不用下载和安装，在网络畅通的情况下可以直接加载运行。

HTML 5 将会模糊 Web 应用与桌面应用的界限，特别是对于分布式的应用，HTML 5 有着先天的优势，不论是开发还是部署都没有那么麻烦。

HTML 5 就是为移动而生的。乔布斯的苹果重新定义了移动互联网，HTML 5 有望成为第二个苹果。HTML 5 让移动云也成为了可能。

浏览器即操作系统。沿着 HTML 5 的发展思路，越来越多的主流应用就是基于浏览器运行的 Web 应用，传统的桌面应用不再是主流，浏览器也将会成为软件运行的主要环境。对此，Google 已经开发了 Chrome 操作系统，该操作系统的目标是包含丰富的基于 HTML API 实现的功能，它提供完美的用户体验，同时使其上运行的应用程序完全符合标准的 Web 体系架构。

1.4　小　　结

本章全面介绍了 HTML 5 的相关发展情况。首先回顾了 HTML 的发展历史，重点介绍了 HTML 5 的设计理念、新增的原生功能及其带来的好处；在对于 HTML 5 的新功能介绍中，重点介绍了新增的一些重要元素、全新的选择器功能和 HTML 5 的开发调试；最后展望了 HTML 5 的未来。本章的难点在于对两个时间点的理解，以及对设计理念和新增原生功能的理解，因为这些方面容易造成误解，对于初始接触 HTML 5 的读者来说，一定要对 HTML 5 有着正确的认识。

下一章，我们将介绍一下 CSS 3 层叠样式表。

1.5　习　　题

【习题 1】列出 HTML 5 发展的两个时间点，并说明发展目标。

【习题 2】在向下兼容方面，HTML 5 有着很强的兼容能力，允许存在不严谨的写法。这一设计理念遵循的是什么法则？

【习题 3】列出 HTML 5 新增的原生功能（至少列出 4 个）和新增的标签元素（至少列出 6 个）。

【习题 4】列出 HTML 5 新增的 3 个选择器函数。

【习题 5】尝试把一个 window 对象输出到控制台。

第 2 章　CSS 3 层叠样式表

说起 HTML 5，必然会提到 CSS 3，不禁让人想起圆角、阴影、渐变页面渲染效果。CSS 3 是 CSS 技术的升级版本，是下一代样式表语言。CSS 3 语言开发是朝着模块化发展的。以前的规范作为一个模块实在是太庞大而且比较复杂，所以把它分解为一些小的模块，更多新的模块也被加入进来。这些模块包括：盒子模型、列表模块、超链接方式、语言模块、背景和边框、文字特效、多栏布局等。本章将对 CSS 3 进行全面的概要的介绍，让大家对 CSS 3 有一个初步的、总体的认识。

2.1　CSS 3 简介

首先，我们来了解一下 CSS 3 的背景。

2.1.1　CSS 3 的历史背景

CSS 的发展是伴随着 HTML 的发展而发展的。

从 1990 年代初 HTML 被发明开始，样式表就以各种形式出现了。不同的浏览器结合了它们各自的样式语言，读者可以使用这些样式语言来调节网页的显示方式。一开始样式表是给读者用的，最初的 HTML 版本只含有很少的显示属性，读者来决定网页应该怎样被显示。

但随着 HTML 的成长，为了满足设计师的要求，HTML 获得了很多显示功能。随着这些功能的增加，额外的样式语言变得越来越没有意义了。

1994 年哈坤・利提出了 CSS 的最初建议。伯特・波斯（Bert Bos）当时正在设计一个叫做 Argo 的浏览器，他们决定一起合作设计 CSS。

当时已经有过一些样式表语言的建议了，但 CSS 是第一个含有“层叠”的主意的。在 CSS 中，一个文件的样式可以从其他的样式表中继承下来。读者在有些地方可以使用他自己更喜欢的样式，在其他地方则继承，或“层叠”作者的样式。这种层叠的方式使作者和读者都可以灵活地加入自己的设计，混合各人的爱好。

哈坤于 1994 年在芝加哥的一次会议上第一次展示了 CSS 的建议，1995 年他与波斯一起再次展示这个建议。当时 W3C 刚刚创建，W3C 对 CSS 的发展很感兴趣，它为此组织了一次讨论会。哈坤、波斯和其他一些人（如微软的托马斯・雷尔登）是这个项目的主要技术负责人。

- 1996 年底，CSS 已经完成。同年 12 月发 CSS 发布了第一个版本的规范 CSS1。
- 1997 年初，W3C 内组织了专门管 CSS 的工作组，其负责人是克里斯・里雷。这

个工作组开始讨论第一版中没有涉及的问题。

- 1998 年 5 月发布了第二个版本的规范 CSS 2。
- 2002 年工作组又启动了对 CSS 2.1 的开发，这是 CSS 2 的修订版，最终于 2004 年正式发布了修订版本 CSS 2.1。CSS 2.1 是目前最流行、浏览器支持最完整的版本。

尽管 CSS 3 的开发工作在 2000 年以前就启动了，但距最终的发布还有相当长的时间。为了提高开发速度以及使各个浏览器能够渐进地支持它，CSS 3 采用模块化的开发方案，每个模块都能独立地实现和发布，这也为未来的 CSS 扩展奠定了基础。

2.1.2　CSS 3 的发展现状

目前，CSS 3 规范尚处于完善之中，因此浏览器的支持程度各有不同。为了让用户能够体验到 CSS 3 的好处，各主流浏览器都定义了自己的私有属性。

1. 模块化的发展

CSS 3 开始遵循模块化的开发。以前的规范作为一个模块实在是太庞大而且比较复杂，所以，CSS 3 把它分解为多个小的模块。这样，有助于理清各个模块规范之间的关系。

CSS 3 的模块化规范，显得非常灵活。一个 CSS 规范如果要完整地获得浏览器的支持，是非常困难的，但是浏览器选择完整支持某个模块的规范是比较容易实现的。反过来，如果要衡量一个浏览器对 CSS 3 的支持程度，可以各个模块分别衡量。

CSS 3 模块化的发展有利于未来的扩展。当 CSS 需要增加新的规范时，非常不希望其他规范也跟着变动。模块化的发展，使得每个独立的模块都能根据需要进行独立的更新。当增加新的特性或模块时，不会影响已经存在的特性。如表 2.1 所示为 CSS 3 中的模块。

表 2.1　CSS 3 中的模块

模 块 名 称	功 能 描 述
Basic box model	定义各种与盒相关的样式
Line	定义各种与直线相关的样式
Lists	定义各种与列表相关的样式
Hyperlink Presentation	定义各种与超链接相关的样式。例如锚的显示方式、激活时的视觉效果等
Presentation Levels	定义页面中元素的不同的样式级别
Speech	定义各种与语音相关的样式。例如音量、音速、说话间歇时间等属性
Background and border	定义各种与背景和边框相关的样式
Text	定义各种与文字相关的样式
Color	定义各种与颜色相关的样式
Font	定义各种与字体相关的样式
Paged Media	定义各种页眉、页脚、页数等页面元数据的样式
Cascading and inheritance	定义怎样对属性进行赋值
Value and Units	将页面上各种各样的值与单位进行统一定义，以供其他模块使用
Image Values	定义对 image 元素的赋值方式
2D Transforms	在页面中实现二维空间上的变形效果

续表

模 块 名 称	功 能 描 述
3D Transforms	在页面中实现三维空间上的变形效果
Transforms	在页面中实现平滑过渡的视觉效果
Animations	在页面中实现动画
CSSOM View	查看管理页面或页面的视觉效果，处理元素的位置信息
Syntax	定义 CSS 样式表的基本结构、样式表中的一些语法细节、浏览器对于样式表的分析规则
Generated and Replaced Content	定义怎样在元素中插入内容
Marquee	定义当一些元素的内容太大，超出了指定的元素尺寸时，是否以及怎样显示溢出部分
Ruby	定义页面中 ruby 元素（用于显示拼音文字）的样式
Writing Modes	定义页面中文本数据的布局方式
Basic User Interface	定义在屏幕、纸张上进行输出时页面的渲染方式
Namespaces	定义使用命名空间时的语法
Media Queries	根据媒体类型来实现不同的样式
'Reader' Media Type	定义用于屏幕阅读器之类的阅读程序时的样式
Multi-column Layout	在页面中使用多栏布局方式
Template Layout	在页面中使用特殊布局方式
Flexible Box Layout	创建自适应浏览器窗口的流动布局或自适应字体大小的弹性布局
Grid Position	在页面中使用风格布局方式
Generated Content for Paged Media	在页面中使用印刷时使用的布局方式

2. 浏览器支持情况

尽管 CSS 3 的很多新的特性很受开发者的欢迎，但并不是所有的浏览器都支持它。各个主流浏览器都定义了各自的私有属性，以便能够让用户体验 CSS 3 的新特性。

私有属性固然可以避免不同浏览器中解析同一个属性时出现冲突，但是也给设计师们带来诸多不便，需要编写更多的 CSS 代码，而且也没有解决同一页面在不同浏览器中表现不一致的问题。

尽管私有属性有很多弊端，但是也为设计师们提供了较大的选择空间，至少在 CSS 3 规范发布以前，能表现一些特定的 CSS 3 的效果。

Webkit 引擎的浏览器（如 Safari、Chrome）的私有属性的前缀是-webkit-；Gecko 引擎的浏览器（如 Firefox）的私有属性的前缀是-moz-；Opera 浏览器的私有属性的前缀是-o-；IE 浏览器（限于 IE8+）的私有属性的前缀是-ms-。

2.1.3　CSS 3 新特性预览

与之前的版本相比，CSS 3 的改进是非常大的。CSS 3 不仅仅进行了修订和完善，更增加了很多新的特性，把样式表的功能发挥得淋漓尽致。之前的很多效果都借助图片和脚本来实现，现在只需要几行代码就能搞定了。这不仅简化了设计师的工作，页面代码也更

加简洁和清晰。

下面浏览一下 CSS 3 的主要新特性。

1. 功能强大的选择器

CSS 3 增加了更多的 CSS 选择器，可以实现更简单但是更强大的功能。例如，在属性选择符中引入通配符、灵活的伪类选择符 nth-child()等。

2. 文字效果

在 CSS 3 中，可以给文字增加阴影、描边和发光等效果；还可以自定义特殊的字体，如艺术字体等网络上不常用的字体。

3. 边框

在 CSS 3 中，可以直接给边框设计圆角、阴影、边框背景等，其中边框背景会自动把背景图切割显示。

4. 背景

背景图片的设计更加灵活：不但可以改变背景图片的大小、裁剪背景图片，还可以设置多重背景。

5. 色彩模式

CSS 3 的色彩模式，除了支持 RGB 颜色外，还支持 HSL（色调、饱和度、亮度），并且针对这两种色彩模式，又增加了可以控制透明度的色彩模式。以后，再设计半透明效果就没有那么麻烦了。

6. 盒布局和多列布局

在布局方面，CSS 3 新增了两种布局方式：灵活的盒布局和多列布局。这两种布局，可以弥补现有布局中的不足，为页面布局提供了更多的手段，并大幅度地缩减了代码。

7. 渐变

CSS 3 已经支持渐变的设计。这样，不但告别切图的时代，而且设计也更加灵活，后期的维护也极为方便。

8. 动画

有了 CSS 3 的动画，设计师们不用编写脚本，直接就可以让页面元素动起来，并且不会影响整体的页面布局。

9. 媒体查询

CSS 3 提供了丰富的媒体查询功能，可以根据不同的设备、不同的屏幕尺寸来自动调整页面的布局。

2.2 增强的选择器功能

在样式表中，选择器是一个非常重要的功能。伴随 CSS 3 和 HTML 5 的发展，选择器的功能已经超出了 CSS 的应用范围，发展成为一个独立的选择器规范，目前还可以应用于脚本的选择查询。针对样式表选择器，CSS 3 新增了诸多选择符，并兼容 CSS 1 和 CSS 2 中的选择符。

下面系统地介绍一下 CSS 选择符，并详细介绍 CSS 3 新增的选择符。

2.2.1 元素选择符和关系选择符

元素选择符和关系选择符是 CSS 中最基本的选择符。如表 2.2 所示为元素选择符；如表 2.3 所示为关系选择符。

表 2.2 CSS元素选择符

选择符	名 称	简 介	版本
*	通配选择符	匹配所有元素	CSS 2
E	类型选择符	匹配指定类型的元素	CSS 1
E#myid	ID 选择符	匹配唯一标识 id 属性等于 myid 的 E 元素	CSS 1
E.myclass	类选择符	匹配 class 属性值为 myclass 的所有 E 元素	CSS 1

表 2.3 CSS关系选择符

选择符	名 称	简 介	版本
E F	包含选择符	选择所有被 E 元素包含的 F 元素	CSS 1
E,F,G	选择符分组	选择所有的 E 元素、F 元素和 G 元素	CSS 1
E>F	子对象选择符	选择所有作为 E 元素的子元素 F	CSS 2
E+F	相邻选择符	选择紧贴在 E 元素之后 F 元素	CSS 2

元素选择符和关系选择符在 CSS 3 之前已经非常完善，并且使用频率也较高，所以在 CSS 3 中完全沿用之前的版本，暂时没有再提供新的选择符。

2.2.2 属性选择符

在 CSS 3 中，属性选择符已经构成了非常强大的标签属性过滤体系，如表 2.4 所示。

表 2.4 CSS属性选择符

选 择 符	简 介	版本
E[attr]	选择具有 attr 属性的 E 元素	CSS 2
E[attr="val"]	选择具有 attr 属性且属性值等于 value 的 E 元素	CSS 2
E[attr~="val"]	选择具有 attr 属性且属性值为一个用空格分隔的字词列表，其中一个等于 val 的 E 元素	CSS 2

续表

选　择　符	简　　介	版本
E[attr|="val"]	选择具有 attr 属性且属性值为一个用连字符分隔的字词列表，由 val 开始的 E 元素	CSS 2
E[attr^="val"]	选择具有 attr 属性且属性值为以 val 开头的字符串的 E 元素	CSS 3
E[attr$="val"]	选择具有 attr 属性且属性值为以 val 结尾的字符串的 E 元素	CSS 3
E[attr*="val"]	选择具有 attr 属性且属性值为包含 val 的字符串的 E 元素	CSS 3

由表 2.4 可知，CSS 3 新增了 3 个属性选择符：E[attr^="val"]、E[attr$="val"]、E[attr*="val"]。这些选择符遵循了惯用的编码规则，选用了^、$和*这三个通用的匹配运算符，具有如下意义。

- ^表示匹配起始符。
- $表示匹配结束符。
- *表示匹配任意字符。

下面我们通过一个示例来了解这 3 个属性选择符的使用方法。

1. 属性选择符E[attr^="val"]

该选择符具有 attr 属性且属性值为一个用空格分隔的字词列表，其中一个等于 val 的 E 元素。

【示例 2-1】 CSS 属性选择符。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS 属性选择符</title>
<style type="text/css">
ul{
    margin:0;
    padding:10px 10px 10px 30px;
}
li{
    margin:1px;
    font-family:Courier New;
    font-weight:bold;
}
/* 属性选择符 E[attr^="val"] */
li[lang^="a"]{
    background-color:#F60;
}
</style>
</head>
<body>
<ul>
  <li lang="af">lang=af aaaaaaaaa</li>
  <li lang="ar">lang=ar bbbbbbbbb</li>
  <li lang="be">lang=be ccccccccc</li>
  <li lang="bg">lang=bg ddddddddd</li>
  <li lang="br">lang=br eeeeeeeee</li>
  <li lang="ca">lang=ca ffffffff</li>
  <li lang="cs">lang=cs gggggggg</li>
  <li lang="da">lang=da hhhhhhhh</li>
```

```
  <li lang="de">lang=de iiiiiiiii</li>
</ul>
</body>
</html>
```

运行结果如图 2-1 所示。

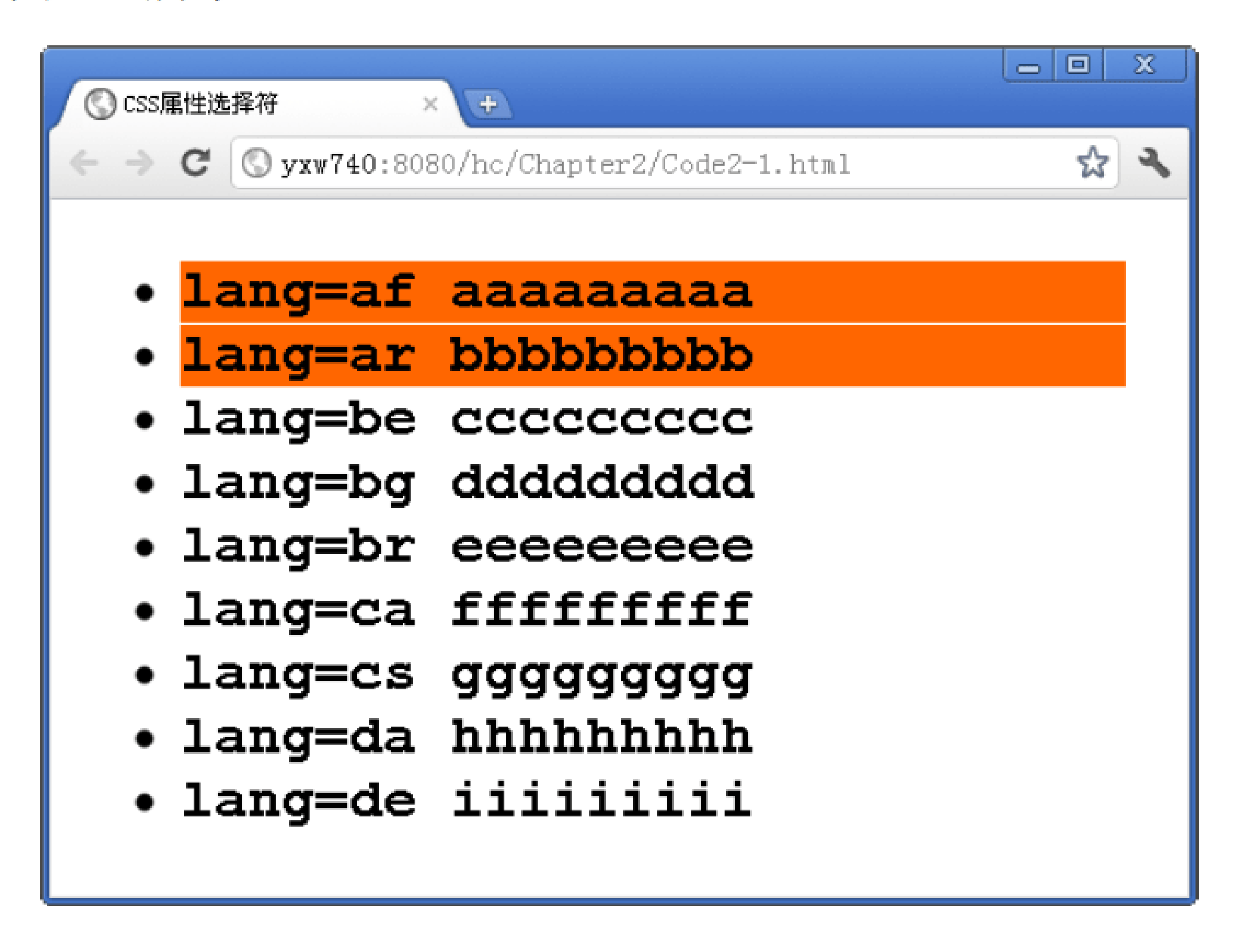


图 2-1　属性选择符 E[attr^="val"]

由图 2-1 所示的运行结果可知，li[lang^="a"]匹配了属性 lang 的值是以"a"开头的 li 元素。

2. 属性选择符E[attr$="val"]

如果使用的是属性选择符 E[attr$="val"]，关键样式代码修改如下：

```
li[lang$="a"]{
    background-color:#F60;
}
```

运行结果如图 2-2 所示。

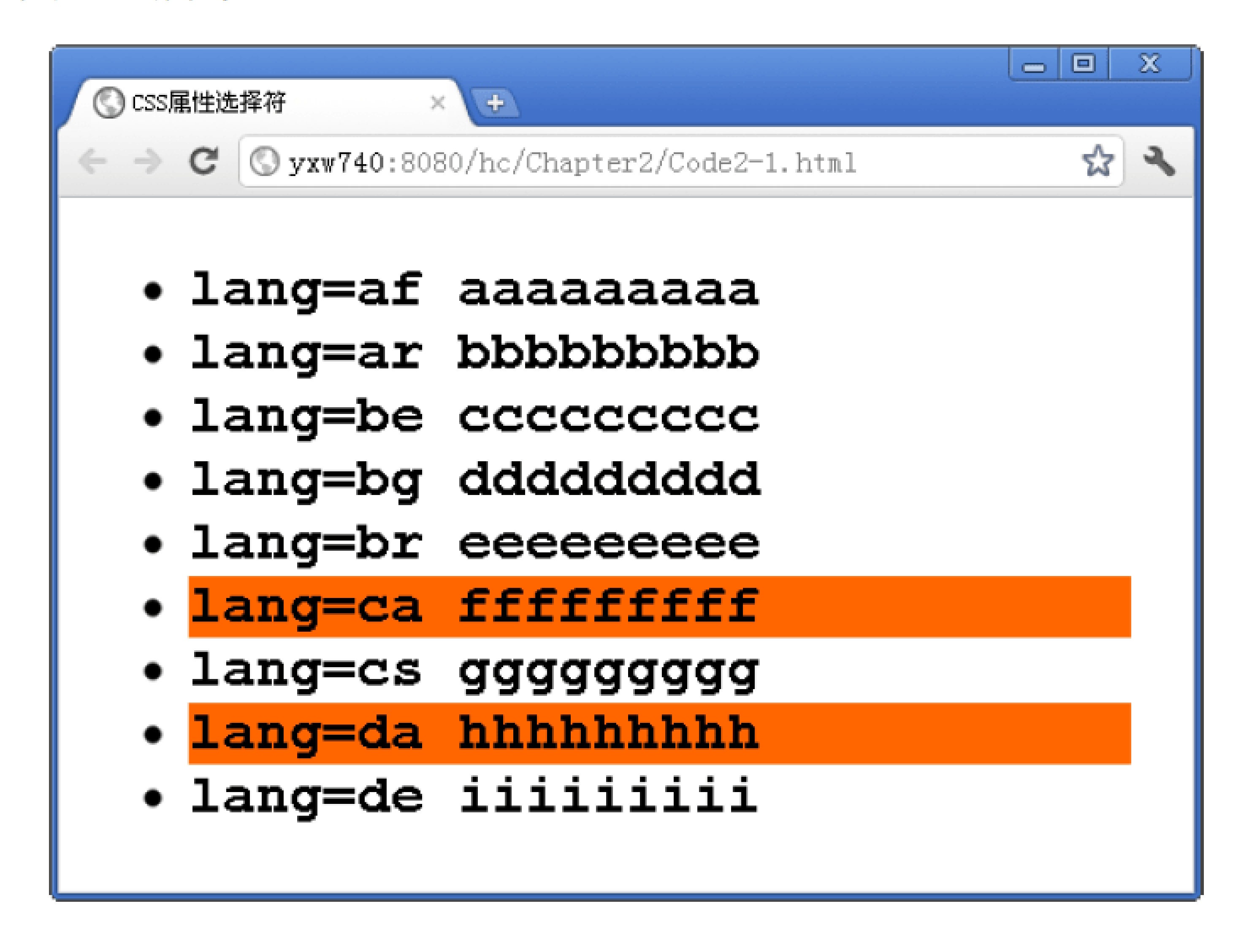


图 2-2　属性选择符 E[attr$="val"]

由图 2-2 所示的运行结果可知，li[lang$="a"]匹配了属性 lang 的值是以"a"结尾的 li 元素。

3. 属性选择符E[attr*="val"]

如果使用的是属性选择符 E[attr*="val"]，关键样式代码修改如下：

```
li[lang*="a"]{
    background-color:#F60;
}
```

运行结果如图 2-3 所示。

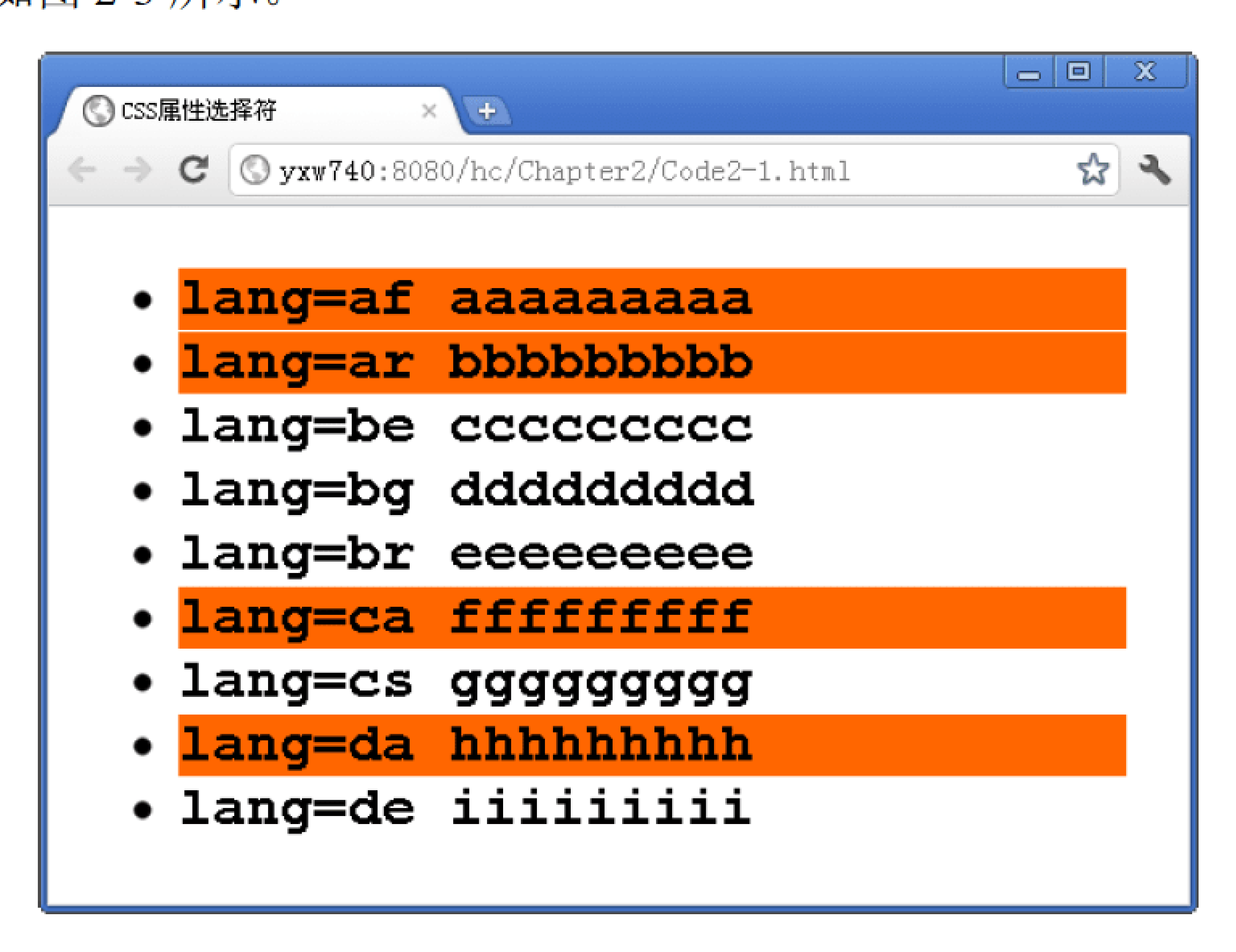


图 2-3　属性选择符 E[attr*="val"]

由图 2-3 所示的运行结果可知，li[lang*="a"]匹配了属性 lang 的值含有字符"a"的 li 元素。

2.2.3　结构伪类选择符

CSS 3 新增的结构伪类选择符，可以通过文档结构的相互关系来匹配特定的元素。对于有规律的文档结构，可以减少 class 属性和 id 属性的定义，使得文档结构更加简洁。结构伪类选择符如表 2.5 所示。

表 2.5　CSS结构伪类选择符

选　择　符	简　　介	版本
E:root	选择匹配 E 所在文档的根元素	CSS 3
E:not(s)	选择匹配所有不匹配简单选择符 s 的 E 元素	CSS 3
E:empty	匹配没有任何子元素（包括 text 节点）的元素 E	CSS 3
E:target	匹配当前链接地址指向的 E 元素	CSS 3
E:first-child	匹配父元素的第一个子元素 E	CSS 2
E:last-child	匹配父元素的最后一个子元素 E	CSS 3
E:nth-child(n)	匹配父元素的第 n 个子元素 E	CSS 3

续表

选　择　符	简　　介	版本
E:nth-last-child(n)	匹配父元素的倒数第 *n* 个子元素 E	CSS 3
E:only-child	匹配父元素仅有的一个子元素 E	CSS 3
E:first-of-type	匹配同类型中的第一个同级兄弟元素 E	CSS 3
E:last-of-type	匹配同类型中的最后一个同级兄弟元素 E	CSS 3
E:only-of-type	匹配同类型中的唯一一个同级兄弟元素 E	CSS 3
E:nth-of-type(n)	匹配同类型中的第 *n* 个同级兄弟元素 E	CSS 3
E:nth-last-of-type(n)	匹配同类型中的倒数第 *n* 个同级兄弟元素 E	CSS 3

1. 选择符root、not、empty和target

- 选择符 E:root：选择匹配 E 所在文档的根元素。所谓根元素就是位于文档结构中的顶层元素。在 HTML 页面中，根元素就是 html 元素，此时该选择符与 html 类型选择符匹配的内容相同。
- 选择符 E:not(s)：选择匹配所有不匹配简单选择符 s 的 E 元素。
- 选择符 E:empty：选择匹配 E 的元素，且该元素不包含子节点。文本也属于节点。
- 选择符 E:target：选择匹配当前链接地址指向的 E 元素。

2. 选择符first-child、last-child、nth-child和nth-last-child

利用这几个选择符，可以指定一个父元素中的第一个子元素、最后一个子元素、指定序号的子元素、第偶数个子元素和第奇数个子元素。

- 选择符 E:first-child：匹配父元素的第一个子元素。
- 选择符 E:last-child：匹配父元素的最后一个子元素。
- 选择符 E:nth-child(n)：匹配父元素中第 *n* 个位置的子元素。其中，参数 *n* 可以是一个数字、关键字（odd、even）、公式（2*n*、2*n*+1 等）。参数 *n* 的索引起始值为 1，而不是 0。使用方法例如：tr:nth-child(5)匹配表格里的第三个 tr 元素；tr:nth-child(2n)和 tr:nth-child(even)匹配表格里的第偶数个 tr 元素；tr:nth-child(2n+1) 和 tr:nth-child(odd)匹配表格里第奇数个 tr 元素。
- 选择符 E:nth-last-child(n)：匹配父元素中倒数第 *n* 个位置的子元素。与选择符 E:nth-child(n)的计算顺序相反，语法和用法均相同。

下面我们通过设计一个表格来了解这四个选择符。表格的第一行为标题行，最后一行为翻页预留，其他行则交替显示两种不同的样式。

【示例 2-2】 使用伪类选择符设计表格。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS 属性选择符</title>
<style type="text/css">
table {
    border-collapse:collapse;
    font-size:12px;
}
```

```
table td {
    font-size:12px;
    padding:0 3px;
    line-height:22px;
}
tr:nth-child(even) {
 background:#e6e6e6;
}
tr:first-child {
    background:-webkit-gradient(linear, left top, left bottom, from
    (#dbdbdb), to(#cccccc));
    background:-moz-linear-gradient(left, #dbdbdb, #cccccc);
}
tr:last-child {
    background:#d6d6d6;
}
</style>
</head>
<body>
数据表格
<table  width="100%"  border="1"  bordercolor="#CCCCCC"  cellpadding="0"
cellspacing="0">
  <tr>
    <td>1 编号</td> <td>标题</td> <td>作者</td> <td>时间</td>
  </tr>
  <tr>
    <td>2</td> <td> </td> <td> </td> <td> </td>
  </tr>
  ……
</table>
</body>
</html>
```

运行结果如图 2-4 所示。



图 2-4　设计的表格

在示例 2-2 中，首先使用 tr:nth-child(even)来控制偶数行的背景颜色加深；接着使用 tr:first-child 设置了第一行的样式；最后使用 tr:last-child 设置了最后一行的样式。在本示例

中，最后一行也可以使用 tr:nth-last-child(1)来替代。

3. 选择符nth-of-type和nth-last-of-type

- ❑ 选择符 E: nth-of-type (n)：匹配父元素中第 *n* 个子元素 E。参数 *n* 的使用与 E:nth-child(n)相同。不同的是，在匹配父元素的子元素时，首先把匹配 E 的子元素筛选出来单独排序，非 E 的子元素不参与排序。
- ❑ 选择符 E:nth-last- of-type (n)：匹配父元素中倒数第 *n* 个子元素。与选择符 E:nth-of-type (n)的计算顺序相反，语法和用法均相同。

4. 选择符only-child、first-of-type、last-of-type和only-of-type

- ❑ 选择符 E: first-of-type：匹配父元素中第一个子元素 E。等同于 E: nth-of-type (1)的匹配效果。
- ❑ 选择符 E: last-of-type：匹配父元素中最后一个子元素 E。等同于 E: nth-last-of-type (1)的匹配效果。
- ❑ 选择符 E: only-child：匹配父元素中唯一子元素 E。父元素只能有唯一的子元素，并且该元素必须为元素 E，才能匹配成功。
- ❑ 选择符 E: only-of-type：匹配父元素中只包含一个子元素 E。父元素可以有多个子元素，但子元素 E 只能有一个才能匹配成功。

2.2.4　UI 元素状态伪类选择符

在 CSS 3 中，还有一种伪类选择符叫 UI 元素状态伪类选择符，可以设置元素处在某种状态下的样式，在人机交互过程中，只要元素的状态发生了变化，选择符就有可能会匹配成功。下面列出了 CSS 中的 UI 元素状态伪类选择符，如表 2.6 所示。

表 2.6　CSS UI元素状态伪类选择符

选择符	简　　介	版　　本
E:link	设置超链接 a 在未被访问前的样式	CSS 1
E:visited	设置超链接 a 在其链接地址已被访问过时的样式	CSS 1
E:hover	设置元素在其鼠标悬停时的样式	CSS 1/CSS 2
E:actvie	设置元素在被用户激活（在鼠标单击与释放之间发生的事件）时的样式	CSS 1/CSS 2
E:focus	设置元素在成为输入焦点（该元素的 onfocus 事件发生）时的样式	CSS 1/CSS 2
E:checked	匹配所有用户界面（form 表单）上处于选中状态的元素 E（用于 input type 为 radio 与 checkbox 时）	CSS 3
E:enabled	匹配所有用户界面（form 表单）上处于可用状态的元素 E	CSS 3
E:disabled	匹配所有用户界面（form 表单）上处于禁用状态的元素 E	CSS 3

由表 2.6 可知，CSS 3 新增了 3 个 UI 元素状态伪类选择符：E:checked、E:enabled、E:disabled。这些选择符具有如下的意义。

- ❑ 选择符 E: checked：匹配所有用户界面（form 表单）上处于选中状态的元素 E。
- ❑ 选择符 E: enabled：匹配所有用户界面（form 表单）上处于可用状态的元素 E。

- 选择符 E: disabled：匹配所有用户界面（form 表单）上处于禁用状态的元素 E。

这 3 个 UI 元素状态伪类选择符，主要应用于表单的设计，可以设计出交互性更强、更具人性化的表单 UI 界面。

2.2.5　伪元素选择符

在 CSS 3 中，还有一种伪元素选择符，它并不是针对真正的元素使用的选择符，而是针对 CSS 已经定义好的伪元素使用的选择符。下面是 CSS 3 中改进过的伪元素选择符，如表 2.7 所示。

表 2.7　CSS伪元素选择符

选　择　符	简　　介	版本
E:first-letter/E::first-letter	设置对象内的第一个字符的样式	CSS 1/CSS 3
E:first-line/E::first-line	设置对象内的第一行的样式	CSS 1/CSS 3
E:before/E::before	设置在对象前（依据对象树的逻辑结构）发生的内容。用来和 content 属性一起使用	CSS 2/CSS 3
E:after/E::after	设置在对象后（依据对象树的逻辑结构）发生的内容。用来和 content 属性一起使用	CSS 2/CSS 3
E::selection	设置对象被选择时的颜色	CSS 3

由表 2.7 可知，CSS 3 的伪元素选择符中的冒号都改成了双冒号。伪元素选择符的使用方法如下：

```
选择符::伪元素{属性:值}
```

下面我们通过一个示例来介绍伪元素选择符的使用方法。

【示例 2-3】 使用伪元素选择符。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>添加 content 内容</title>
<style type="text/css">
/* 突出第一个字 */
p:first-letter {
    font-size:24px;
    font-weight:bold;
}
/* 链接前加图片 */
a[href$=doc]::before {
 content:url(images/doc.png);
}
a[href$=pdf]::before {
 content:url(images/pdf.png);
}
</style>
</head>
<body>
<p>以下是参考资料：<br>
  <a href="images/test.doc">参考资料 1</a><br>
```

```
  <a href="images/test.pdf">参考资料 2</a> </p>
</body>
</html>
```

运行结果如图 2-5 所示。

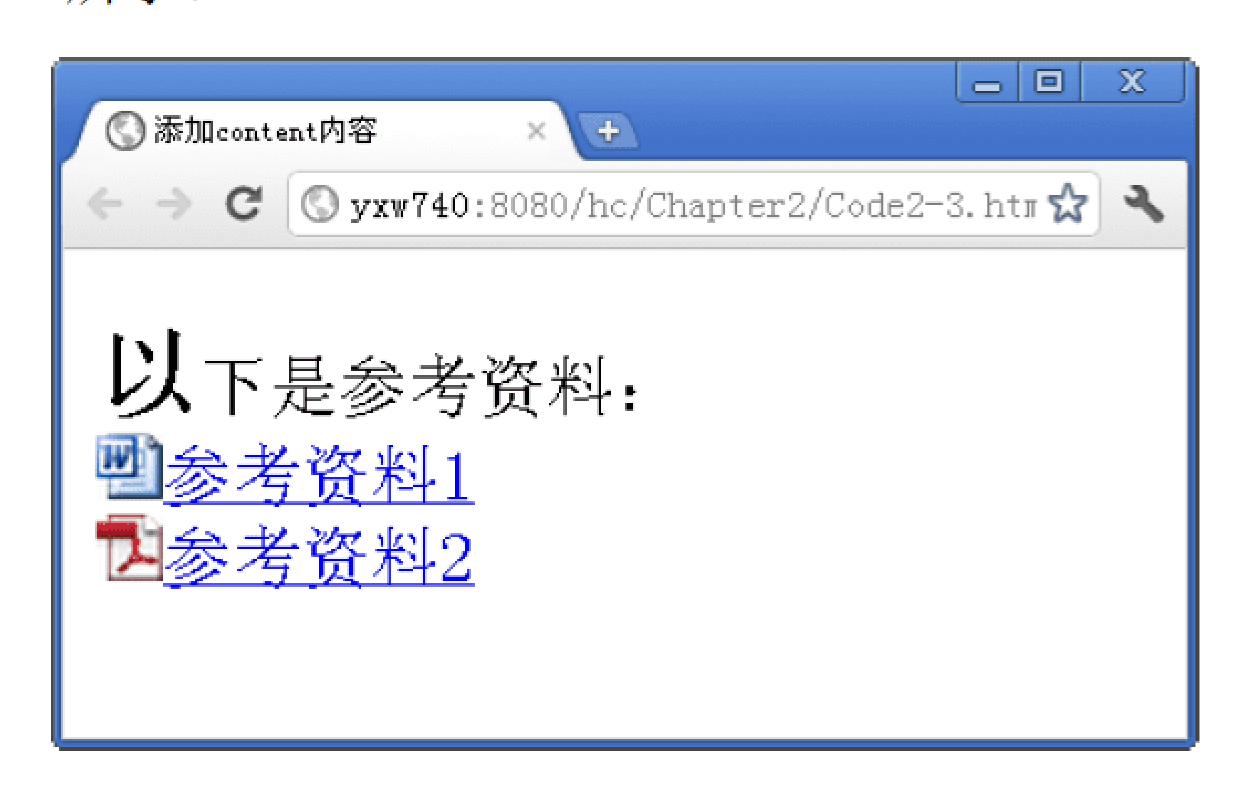


图 2-5　使用伪元素选择符

由图 2-5 可以看出，链接前面的图片是通过在样式表中使用伪元素 before 来实现的；首个字符是使用伪元素 first-letter 来实现的。其他伪元素的使用方法是一致的。

2.3　小　　结

本章主要回顾了 CSS 的发展历程及目前的发展状况，并简要介绍了 CSS 3 的新特性。其中重点介绍了 CSS 3 的选择器功能，为设计样式表提供更加灵活的代码设计方案。本章的难点在于对选择器功能的掌握。新增的选择器功能包括属性选择符、伪类选择符和伪元素选择符，它们在选择文档元素方面显得非常灵活，作为前端开发人员，一定要掌握它们。

下一章将进入 CSS 3 学习的第一步——文本、背景、边框不再单调。

2.4　习　　题

【习题 1】CSS 3 遵循的是什么样的发展方式？描述这种发展的益处。

【习题 2】列举一下 CSS 3 为哪些方面提供了新的特性（至少列出 4 个方面）？描述这些新特性。

【习题 3】列举 CSS 3 新增的三个属性选择符，并描述其含义。

第 2 篇　基于 CSS 3 的 Web 界面设计实战

第 3 章　文本、背景、边框不再单调

在制作网页时，经常会围绕文本、背景和边框几个方面进行样式表设置。但在 CSS 2.0 中，如果涉及如阴影、圆角、多重背景等效果，常常因为不能实现，转而寻求其他办法。在颜色选择及半透明颜色使用方面也极为不便。这些问题，在 CSS 3 中都有便捷的解决方案。CSS 3 在原有版本的基础上，扩充了一些非常实用的属性和颜色方案。本章将就这些扩充的内容进行详细讲解。

3.1　文本与字体

在网页设计中，丰富的文本修饰效果，不但可以增添网页的趣味性，而且看起来更加舒服。在 CSS 3 中，在文本修饰方面，可以增加阴影、描边和发光等效果。在排版方面，可以对溢出及换行进行良好的控制。甚至对于特殊少见的字体，也能在客户端显示良好。下面逐步讲解。

3.1.1　多样化的文本阴影——text-shadow 属性

在网页设计中，常常会通过给文本添加阴影、描边和发光等效果，来实现更加丰富的视觉表现。CSS 3 中的阴影属性 text-shadow，不但可以给文本添加阴影，还可以实现文本的描边和发光效果。

1. 参数说明

阴影属性 text-shadow 的使用语法如下：

```
text-shadow: length || length || opacity || color
```

参数及取值说明如下。

- ❑ length：是由浮点数字和长度单位组成的长度值，可以为负值。两个 length 分别表示阴影在水平方向和垂直方向上相对于文字本身的偏移距离。
- ❑ opacity：是由浮点数字和长度单位组成的长度值，不可以为负值，表示阴影效果模糊作用的距离。该值可以省略，表示模糊作用距离为 0，即没有模糊效果。
- ❑ color：是颜色值，表示阴影的颜色。

文本的阴影、描边和发光等效果，就是这些参数的不同组合的结果。

提示：到目前为止，text-shadow 属性已获得所有的浏览器厂商的新版本浏览器的支持，不过在旧的 IE 8 及以下的版本中，是无法支持该属性的。

2. 文本的阴影效果

先看一下 text-shadow 属性的使用示例，为橘黄色文字设置深灰色阴影。其中阴影在水平和垂直方向上的距离均为 5px，模糊作用距离为 3px，阴影颜色为深灰色。

【示例 3-1】 为文字设置深灰色阴影。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>文字阴影</title>
<style type="text/css">
p {
    font-family:Verdana, Geneva, sans-serif;
    font-weight:bold;
    font-size:36px;
    color:#f90;
    text-shadow:5px 5px 3px #333;                    /* 添加文字阴影 */
}
</style>
<body>
<p>阴影属性<br />
  text-shadow</p>
</body>
</html>
```

运行结果如图 3-1 所示。

代码分析：在示例 3-1 中，为属性 text-shadow 设置了向右下的阴影效果，颜色为深灰色。如果偏移值为负数，表示阴影向左或向上偏移。修改 text-shadow 属性值如下：

```
text-shadow:-5px -5px 3px #00f;
```

运行结果将变成如图 3-2 所示。



图 3-1　向右下方向的阴影



图 3-2　向左上方向的阴影

还可以为阴影属性同时设置两种及两种以上的阴影效果。修改 text-shadow 属性值如下：

```
text-shadow:-5px -5px 3px #00f,
             5px 5px 3px #333;
```

运行结果如图 3-3 所示。

3. 文本的描边效果

利用 text-shadow 属性的特性，同时在上、下、左、右四个方向为文字设置多个阴影，

且不设置模糊作用距离（即默认没有模糊效果），就可以实现文本的描边效果了。

【示例 3-2】 为文字设置描边效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>文字描边</title>
<style type="text/css">
p {
    padding:50px;
    font-family:Verdana, Geneva, sans-serif;
    font-weight:bold;
    font-size:36px;
    background-color:#CCC;
    color:#ddd;
    text-shadow:-1px 0 #333,        /* 向左阴影 */
           0 -1px #333,             /* 向上阴影 */
           1px 0 #333,              /* 向右阴影 */
           0 1px #333;              /* 向下阴影 */
}
</style>
<body>
<p>阴影属性<br />
  text-shadow</p>
</body>
</html>
```

运行结果如图 3-4 所示。



图 3-3　同时有两个阴影



图 3-4　描边的文字

代码分析：在示例 3-2 中，为 text-shadow 属性在四个方向上分别设置 1 个像素的阴影，且没有模糊效果，组合起来就是描边效果了。

当然，为了表现更加丰富，每个方向上的阴影的颜色可以有不同的设置。如果将向左和向上的阴影颜色设置为白色，文字就会有凸起的效果。修改 text-shadow 属性如下。

```
text-shadow:-1px 0 #FFF,            /* 向左阴影 */
       0 -1px #FFF,                 /* 向上阴影 */
       1px 0 #333,                  /* 向右阴影 */
       0 1px #333;                  /* 向下阴影 */
```

运行结果如图 3-5 所示。

如果将向右和向下的阴影颜色设置为白色，文字就会有凹陷的效果。修改 text-shadow 属性如下。

```
text-shadow:-1px 0 #333,            /* 向左阴影 */
        0 -1px #333,                /* 向上阴影 */
        1px 0 #FFF,                 /* 向右阴影 */
        0 1px #FFF;                 /* 向下阴影 */
```

运行结果如图 3-6 所示。



图 3-5　凸起的文字效果

图 3-6　凹陷的文字效果

如果设置凹陷的文字，就把向右和向下的阴影颜色改为白色。使用阴影属性，还可以模拟外发光效果，这里不再说明。

4. 文本的发光效果

也可以利用 text-shadow 属性的特性，不设置水平和垂直的偏移距离，仅设置模糊作用距离，这样就可以通过修改模糊值来实现强度不同的发光效果了。

【示例 3-3】 为文字设置描边效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>文字发光</title>
<style type="text/css">
p {
    padding:50px;
    font-family:Verdana, Geneva, sans-serif;
    font-weight:bold;
    font-size:36px;
    background-color:#333;
    color:#f90;
    text-shadow:0 0 10px #fff;          /* 没有偏移的模糊设置 */
}
</style>
<body>
<p>阴影属性<br />
  text-shadow</p>
</body>
</html>
```

运行结果如图 3-7 所示。

代码分析：在示例 3-3 中，text-shadow 属性在水平和垂直的偏移距离均为 0，仅设置模糊效果，加上深灰色背景的衬托，就有了发光的效果了。



图 3-7　发光的文字

提示：虽然通过 text-shadow 属性的渲染可以让网页变得更加丰富和生动，但不建议整个页面处处都充斥着这样的效果，那样网页会变得凌乱。对于设计良好的网页，如果要包含阴影、描边和发光等效果，则可以通过该属性轻松实现。为了使 text-shadow 属性能兼容多种内核的旧的浏览器，通常会针对不同的浏览器去写：-moz-text-shadow 对应 Gecko 内核的浏览器，如火狐；-webkit-text-shadow 对应 Webkit 内核的浏览器，如 Chrome 和 Safari 等。随着浏览器的版本更新，阴影属性已获得各浏览器最新版本的支持，不需要加前缀。

3.1.2　溢出文本处理——text-overflow 属性

一个布局良好的页面，会限制列表结构的宽度。如果文本过长，则会导致文本溢出，打乱页面的整体布局，需要截断显示。为了显示友好，CSS 3 新增了溢出文本处理的属性 text-overflow。text-overflow 属性的语法如下：

```
text-overflow : clip | ellipsis | ellipsis-word
```

取值说明：值 clip 表示直接裁切溢出的文本；值 ellipsis 表示文本溢出时，显示省略标记（...），省略标记代替最后一个字符；值 ellipsis-word 也表示文本溢出时，显示省略标记（...），与值 ellipsis 不同的是，省略标记代替的是最后一个词。

【示例 3-4】 溢出文本省略标记。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>溢出文本处理</title>
<style type="text/css">
li {
    list-style:none;
    line-height:22px;
    font-size:12px;
    border-bottom:1px solid #ccc;
    width:220px;                    /* 设置宽度 */
    overflow:hidden;                /* 溢出内容设为隐藏 */
    white-space:nowrap;             /* 强制文本单行显示 */
    text-overflow:ellipsis;         /* 设置溢出文本显示为省略标记 */
}
</style>
```

```
<body>
<ul>
  <li>• 分享开发 HTML 5 手机游戏的 5 个注意</li>
  <li>• 四川射洪县动物园一头黑熊翻墙逃走 副县长率队搜捕 M</li>
  <li>• 《牵挂》台湾热播 张国强受宝岛粉丝追捧</li>
  <li>• 黄怒波回应质疑:并非购买冰岛国土而是投资地产</li>
  <li>• 冰岛总统接受专访力挺中国富豪买地</li>
</ul>
</body>
</html>
```

运行结果如图 3-8 所示。

代码分析：示例 3-4 中，仅设置 text-overflow 属性是不够的。必须设置文本外围的宽度、溢出内容为隐藏（overflow:hidden）、强制文本单行显示（white-space:nowrap），这样设置的 text-overflow 属性值 ellipsis 才能显示为省略标记的效果。

提示： 在兼容性方面，Firefox 和早期的 Opera 不支持该属性，其他浏览器均支持。

- 分享开发HTML5手机游戏的5个注意要点
- 四川射洪县动物园一头黑熊翻墙 逃…
- 《牵挂》台湾热播 张国强受宝岛粉…
- 黄怒波回应质疑:并非购买冰岛国土…
- 冰岛总统接受专访力挺中国富豪买地

图 3-8　文本溢出处理

3.1.3　对齐的文字才好看——word-wrap 和 word-break 属性

一个布局很好的页面，常常会因为换行，导致整个页面参差不齐。CSS 3 采用了 IE 发展的 word-wrap 属性和 word-break 属性，这两个属性在 IE 中一直被支持。

1. 边界换行属性word-wrap

word-wrap 属性，设置或检索当前行超过指定容器的边界时是否断开转行。其语法如下：

```
word-wrap : normal | break-word
```

取值说明：值 normal 为默认的连续文本换行，允许内容超出边界；值 break-word 表示内容将在边界内换行，如果需要，词内换行（word-break）也会发生。

【示例 3-5】 网址的边界换行。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>边界换行</title>
<style type="text/css">
p {
    font-family:Verdana, Geneva, sans-serif;
    border:1px solid #CCC;
    padding:10px;
    width:220px;
    font-size:12px;
    word-wrap:normal;           /* 设置换行属性 */
}
```

```
</style>
<body>
<p> CSS3 is completely backwards compatible, so you will not have to change existing designs. Browsers will always support CSS2. http://www.w3schools.com/css3/css3_intro.asp </p>
<p>CSS3 的是完全向后兼容，所以你不会有改变现有的设计。 浏览器将始终支持 CSS2。http://www.w3schools.com/css3/css3_intro.asp</p>
</body>
</html>
```

运行结果如图 3-9 所示。

代码分析：示例 3-5 中，设置 word-wrap 属性值为 normal，当连续的文本（如网址）过长时，会超出边界显示。如果 word-wrap 属性值为 break-word，则网址不会超出边界。修改 word-wrap 属性值如下：

```
word-wrap:break-word;          /* 设置换行属性为 break-word */
```

运行结果如图 3-10 所示。

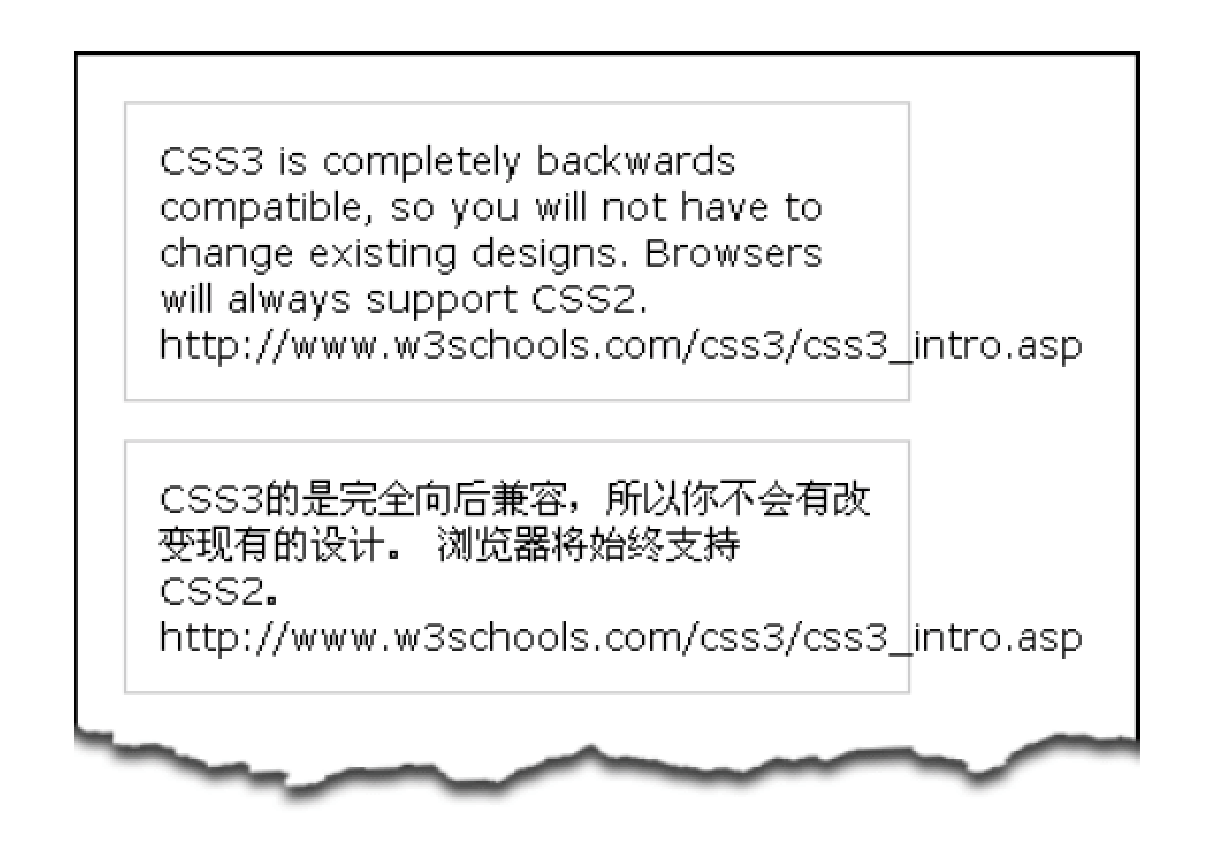


图 3-9　超出边界的网址

图 3-10　边界内换行的网址

2. 字内换行属性word-break

word-break 属性设置或检索对象内文本的字内换行行为，尤其在出现多种语言时。对于中文，应该使用 break-all。其语法如下：

```
word-break : normal | break-all | keep-all
```

取值说明：值 normal，依照亚洲语言和非亚洲语言的文本规则，允许在字内换行；值 break-all，该行为与亚洲语言的 normal 相同，也允许非亚洲语言文本行的任意字内断开，该值适合包含一些非亚洲文本的亚洲文本；值 keep-all，与所有非亚洲语言的 normal 相同。对于中文、韩文、日文，不允许字断开，适合包含少量亚洲文本的非亚洲文本。

该属性的值与使用的语言有关系。下面就通过示例了解其区别。

【示例 3-6】 文字内换行。

```
<!DOCTYPE HTML>
<html>
<head>
```

```
<meta charset="utf-8">
<title>文字内换行</title>
<style type="text/css">
p {
    font-family:Verdana, Geneva, sans-serif;
    border:1px solid #CCC;
    padding:10px;
    width:220px;
    font-size:12px;
    word-break: break-all;          /* 设置换行属性 */
}
</style>
<body>
<p> CSS3 is completely backwards compatible, so you will not have to change
existing designs. Browsers will always support CSS2. http://www.w3schools.
com/css3/css3 intro.asp </p>
<p>CSS3 的是完全向后兼容，所以你不会有改变现有的设计。 浏览器将始终支持 CSS 2。
http://www.w3schools.com/css3/css3_intro.asp</p>
</body>
</html>
```

运行结果如图 3-11 所示。



图 3-11　文字内换行

代码分析：示例 3-6 中，将 word-break 属性值设为 break-all。运行结果如图 3-11 所示，将文字拆分并换行。

如果将属性 word-break 属性值设为 normal，运行结果将如图 3-9 所示。如果将其属性值设为 keep-all，在本示例中，运行结果也将如图 3-9 所示。

3.1.4　使用服务器端的字体——@font-face 规则

在 CSS 的字体样式中，通常会受到客户端的限制，只有在客户端安装了该字体后，样式才能正确显示。如果使用的不是常用的字体，对于没有安装该字体的用户而言，是看不到真正的文字样式的。因此，设计师会避免使用不常用的字体，更不敢使用艺术字体。

为了弥补这个缺陷，CSS 3 新增了字体自定义功能，通过@font-face 规则来引用互联网任一服务器中存在的字体。这样在设计页面的时候，就不会因字体稀缺而受限制。

1. @font-face的语法规则

@font-face 能够加载服务器端的字体文件，让客户端显示客户端所没有安装的字体。其语法规则如下：

```
@font-face: {属性: 取值;}
```

属性及取值如下。

- font-family：设置文本的字体名称。
- font-style：设置文本样式。
- font-variant：设置文本是否为小型大写字母大小写。
- font-weight：设置文本的粗细。
- font-stretch：设置文本是否横向的拉伸变形。
- font-size：设置文本字号大小。
- src：设置自定义字体的相对路径或者绝对路径，可包含 format 信息。此属性只能在@font-face 规则里使用。

其中，font-family 的属性值是用来声明字体名称的，该名称可被当作字体引用。src 也是必要的属性，用于指定字体文件的路径。其他属性，则是选择性使用的。

提示： 对于@font-face 的兼容，主要是字体 format 的问题。因为不同的浏览器对字体格式的支持是不一致的，各种版本的浏览器支持的字体格式有所区别。TureTpe(.ttf)格式的字体对应的 format 属性为“truetype”；OpenType(.otf)格式的字体对应的 format 属性为“opentype”；Embedded Open Type(.eot)格式的字体对应的 format 属性为“eot”。

2. 示例介绍

【示例 3-7】 使用服务器端字体。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>@font-face 规则</title>
<style type="text/css">
@font-face {
    font-family: myfont;                    /* 声明字体名称 */
    src:url(../font/HEMIHEAD.TTF) format("truetype");
                                            /* 指定字体文件路径 */
}
p {
    font-family:myfont;                     /* 使用声明的字体名称定义字体样式 */
    font-size:36px;
    color:#f90;
}
</style>
<body>
<p>Hemi Head</p>
</body>
</html>
```

运行结果如图 3-12 所示。

代码分析：如示例 3-7 所示，在@font-face 的规则里，通过 font-family 属性声明了字体名称 myfont，并通过 src 指定了字体文件的 url 相对地址。在接下来的样式设置中，就可以通过名称 myfont 来引用字体定义的规则了。该示例展示的字体名称为“Hemi Head 426”的字体，效果如图 3-12 所示。



图 3-12　服务器端的字体

通过@font-face 的规则使用服务端字体，为网页设计者们提供了更大的自由空间。服务器端的字体可以根据需要，不受限制地选择，甚至可以选择艺术字体。

提示： 通过@font-face 规则使用服务器字体，不建议应用于中文网站。因为中文的字体文件都是几 MB 到十几 MB，这么大的字体文件，会严重影响网页的加载速度。如果是少量的特殊字体，还是建议使用图片来代替。而英文的字体文件只有几十 KB，非常适合使用@font-face 规则。

如果客户端安装的字体丰富，包含了服务器端提供的字体，出于性能的考虑，我们会尽可能地选择客户端的字体，以避免字体文件的网络传输中造成的性能损失。可以将规则中 src 属性的值通过“local()”来指定本地系统的字体。利用 src 属性可以同时指定多个地址的特性，我们将示例 3-7 修改如下。

【示例 3-8】 同时定义客户端和服务器端字体。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>@font-face 规则</title>
<style type="text/css">
@font-face {
    font-family: myfont;                /* 声明字体名称 */
    src:local("Hemi Head 426"),         /* 指向客户端本地系统字体 */
        url(../font/SUPERSOU.TTF) format("truetype");
                                        /* 指向服务器端的字体文件 */
}
p {
    font-family:myfont;                 /* 使用声明的字体名称定义字体样式 */
    font-size:36px;
    color:#f90;
}
</style>
<body>
<p>@font-face</p>
</body>
</html>
```

运行结果如图 3-13 和图 3-14 所示。



图 3-13　显示客户端字体



图 3-14　显示服务器端字体

代码分析：示例 3-8 中，@font-face 规则里的 src 属性，同时指定了客户端系统中的字体和服务器端提供的字体文件。当客户端存在字体“Hemi Head 426”时，显示结果如图 3-13 所示；当客户端不存在该字体时，则使用服务端提供的字体“Supersoulfighter”，显示结果如图 3-14 所示。本示例为了说明问题，而选择了两种不同的字体。

3.1.5　实验室：丰富的文字样式

在前几节中学习了文本阴影属性，对文本的修饰更加灵活，而使用@font-face 规则，使得选择的字体几乎不受限制。对于服务器提供的字体，也可以使用阴影、颜色、粗体、斜体等样式表进行修饰。文字样式相比之前丰富了很多。下面就结合学过的文字样式，对文字进行多方面修饰。

【示例 3-9】 修饰唐诗《游子吟》。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>唐诗《游子吟》</title>
<style type="text/css">
@font-face {
 font-family: myfont;                          /* 声明字体名称 */
 src:url(../font/maozedong.ttf) format("truetype");
                                               /* 指向服务器端的字体文件 */
}
body {
    padding:0 40px;
}
h1 {
    float:right;
    width:20px;
    margin:0 0 0 10px;
    padding:0;
    font-family:myfont;                        /* 使用声明的字体名称定义字体样式 */
    font-size:33px;                            /* 大小 */
    color:#f90;                                /* 颜色 */
    text-shadow:3px 3px 3px #333;              /* 阴影 */
    word-wrap:break-word;                      /* 边界换行，逗号可在行的开始位置 */
}
p {
    float:right;
    width:20px;
    padding:0;
    margin:0 20px 0 0;
    line-height:33px;
    font-family:myfont;                        /* 使用声明的字体名称定义字体样式 */
    font-size:30px;                            /* 大小 */
    color:#f90;                                /* 颜色 */
    text-shadow:0 0 1px #fff;                  /* 白色阴影，消除锯齿 */
    word-wrap:break-word;                      /* 边界换行，逗号可在行的开始位置 */
}
footer {
    float:right;
```

```
    width:20px;
    padding-top:80px;
    font-family:myfont;                      /* 使用声明的字体名称定义字体样式 */
    font-size:30px;                          /* 大小 */
    color:#f90;                              /* 颜色 */
    text-shadow:0 0 3px #333;                /* 阴影 */
    margin-right:30px;
}
</style>
<body>
<h1>游子吟</h1>
<p>慈母手中线，</p>
<p>游子身上衣。</p>
<p>临行密密缝，</p>
<p>意恐迟迟归。</p>
<p>谁言寸草心，</p>
<p>报得三春晖？</p>
<footer>孟郊</footer>
</body>
</html>
```

运行结果如图 3-15 所示。

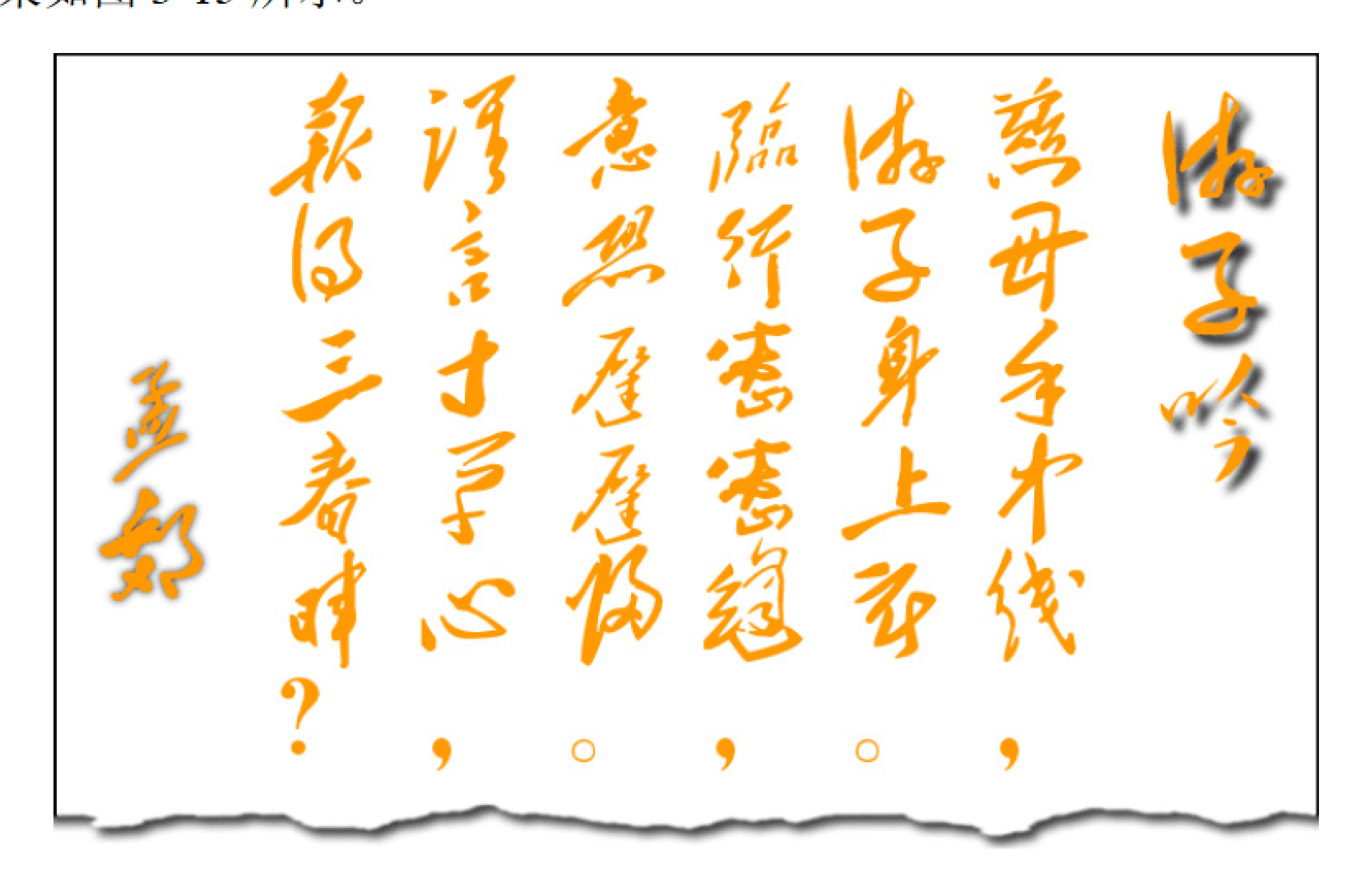


图 3-15　经过修饰的服务器端字体

代码分析：在传统的文字修饰方面，为了突出某个文字，只能使用加粗和斜体，在示例 3-9 中，使用了阴影属性修饰标题为阴影效果，修饰作者名称为发光效果。字体使用的是服务器指定路径下的“草檀斋毛泽东字体”文件。排版方面是模拟的，边界换行属性 word-wrap 值为 break-word，即允许逗号可在行的开始位置。

3.2　色彩模式和不透明度

在 CSS 3 之前，我们通常使用的颜色都属于 RGB 色彩模式，而且颜色本身也不能设置透明度。CSS 3 不但新增了 HSL 色彩模式，还增加了颜色本身的不透明设置和单独的不

透明属性。这两个色彩模式及不透明设置，在整个 HTML 5 框架内都适用。本节就对新增的色彩模式及不透明设置进行讲解。

3.2.1　不再为配色发愁——HSL 色彩模式

HSL 色彩模式是 CSS 3 新增的色彩模式，是工业界的一种颜色标准，通过对色调（Hue）、饱和度（Saturation）、亮度（Lightness）三个颜色通道的变化及它们相互之间的叠加来得到各式各样的颜色。这为颜色和色调的选择提供了充足的余地。这个标准几乎包括了人类视力所能感知的所有颜色，是目前运用最广的颜色系统之一。

1. 参数说明

在 CSS 3 中，HSL 色彩模式的表示语法如下：

```
hsl(<length>, <percentage>, <percentage>)
```

参数及取值说明如下。

- <length>：表示色调（Hue）。衍生于色盘，可以取任意值。其中该值除以 360 所得的余数为 0 表示红色，为 60 表示黄色，为 120 表示绿色，为 180 表示青色，为 240 表示蓝色，为 300 表示洋红色。如图 3-16 所示为色盘模型。
- <percentage>：表示饱和度（Saturation）。表示色调确定的颜色的浓度，即鲜艳程度。值为百分比，范围从 0%到 100%。0%表示灰度，没有颜色；100%说明最鲜艳。
- <percentage>：表示颜色的明亮度（Lightness）。值为百分比，范围从 0%到 100%。0%最暗，50%为均值，100%最亮。

HSL 色彩模式中的色调、饱和度和亮度可用一个圆柱体的空间模型来模拟，圆柱里的每个点都代表一个颜色值。如图 3-17 所示为 HSL 空间模型。

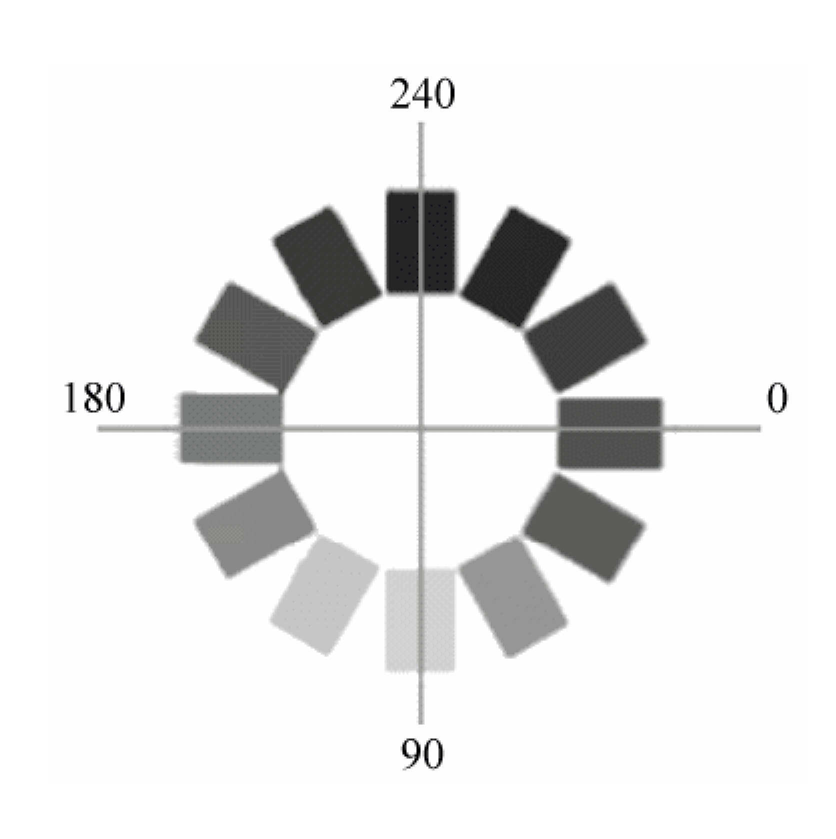


图 3-16　色调的色盘模型

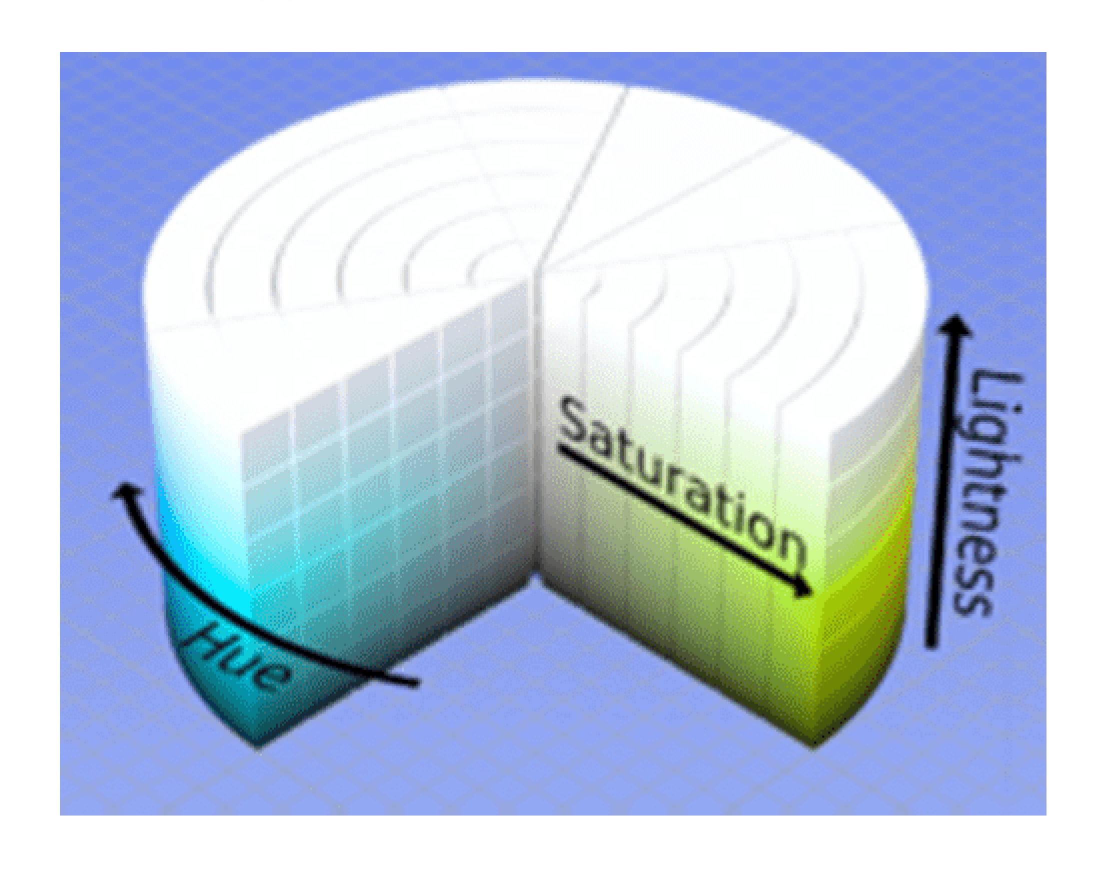


图 3-17　HSL 空间模型

2. 示例介绍

网页设计中的配色也是有规律可循的。利用 HSL 色彩模式，首先确定网页的主色调，即确定 HSL 的色调值；然后通过调整饱和度和亮度，即可在同一色系中选择颜色。这样颜色搭配不会太离谱，整体上也有统一的感觉。

【示例 3-10】 主色调为红色的配色方案表。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>配色方案表</title>
<style type="text/css">
.hsl {
    height:20px;
    border:1px solid #f00;
    padding:10px;
    height:170px;
    background-color:hsl(0, 0%, 90%);        /* 使用 HSL 模式的颜色值 */
    color:hsl(0, 100%, 50%);                 /* 使用 HSL 模式的颜色值 */
    font-size:12px;
    text-align:center;
    line-height:25px;
    width:320px;
}
ul {
    width:320px;
    margin:0;
    padding:10px 0;
    border-top:1px solid #ccc;
}
li {
    float:left;
    margin:1px 0 0 1px;
    width:50px;
    height:15px;
    list-style:none;
    font-size:12px;
    line-height:15px;
}
/* 第一行 */
li:nth-child(8) {background-color:hsl(0, 100%, 100%);}
li:nth-child(9) {background-color:hsl(0, 75%, 100%);}
li:nth-child(10) {background-color:hsl(0, 50%, 100%);}
li:nth-child(11) {background-color:hsl(0, 25%, 100%);}
li:nth-child(12) {background-color:hsl(0, 0%, 100%);}
/* 第二行 */
li:nth-child(14) {background-color:hsl(0, 100%, 75%);}
li:nth-child(15) {background-color:hsl(0, 75%, 75%);}
li:nth-child(16) {background-color:hsl(0, 50%, 75%);}
li:nth-child(17) {background-color:hsl(0, 25%, 75%);}
li:nth-child(18) {background-color:hsl(0, 0%, 75%);}
/* 第三行 */
li:nth-child(20) {background-color:hsl(0, 100%, 50%);}
li:nth-child(21) {background-color:hsl(0, 75%, 50%);}
li:nth-child(22) {background-color:hsl(0, 50%, 50%);}
li:nth-child(23) {background-color:hsl(0, 25%, 50%);}
li:nth-child(24) {background-color:hsl(0, 0%, 50%);}
/* 第四行 */
li:nth-child(26) {background-color:hsl(0, 100%, 25%);}
li:nth-child(27) {background-color:hsl(0, 75%, 25%);}
li:nth-child(28) {background-color:hsl(0, 50%, 25%);}
li:nth-child(29) {background-color:hsl(0, 25%, 25%);}
li:nth-child(30) {background-color:hsl(0, 0%, 25%);}
```

```
/* 第五行 */
li:nth-child(32) {background-color:hsl(0, 100%, 0%);}
li:nth-child(33) {background-color:hsl(0, 75%, 0%);}
li:nth-child(34) {background-color:hsl(0, 50%, 0%);}
li:nth-child(35) {background-color:hsl(0, 25%, 0%);}
li:nth-child(36) {background-color:hsl(0, 0%, 0%);}
</style>
<body>
<div class="hsl">
  <div>色调：0 红色</div>
  <div>竖向：亮度；横向：饱和度</div>
  <ul>
    <li></li> <li>100%</li> <li>73%</li> <li>50%</li> <li>25%</li> <li>0%
    </li>
    <li>100%</li> <li></li> <li></li> <li></li> <li></li> <li></li>
    <li>75%</li> <li></li><li></li> <li></li> <li></li> <li></li>
    <li>50%</li> <li></li> <li></li> <li></li> <li></li> <li></li>
    <li>25%</li> <li></li> <li></li> <li></li> <li></li> <li></li>
    <li>0%</li> <li></li> <li></li> <li></li> <li></li> <li></li>
  </ul>
</div>
</body>
</html>
```

运行结果如图 3-18 所示。

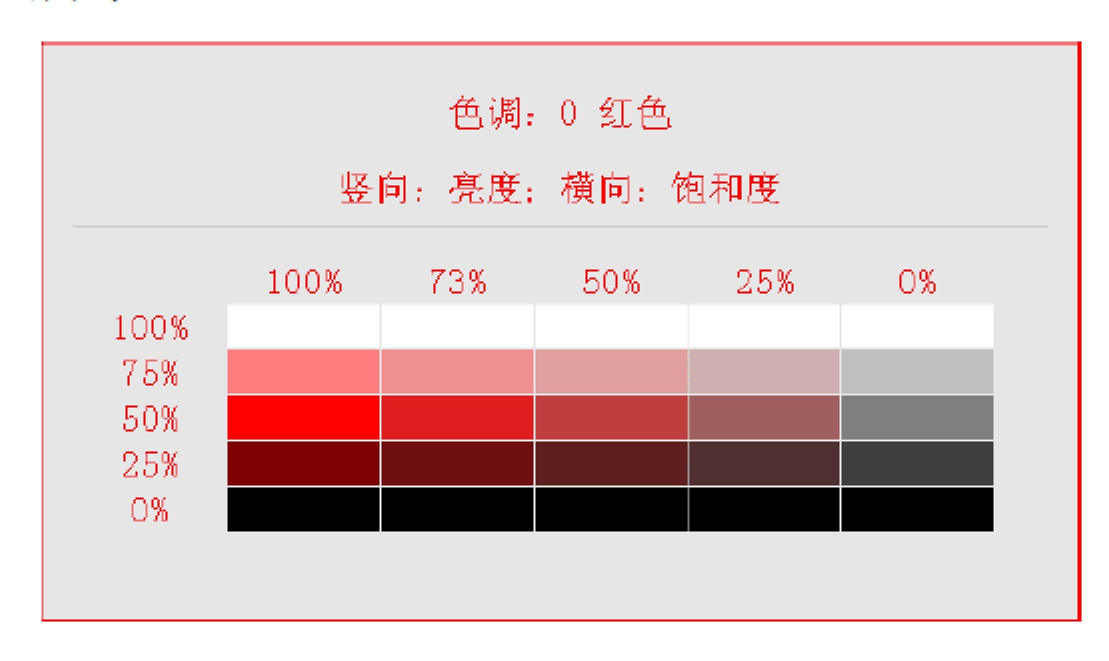


图 3-18　主色调为红色的配色方案表

代码分析：示例 3-10 中，在所有有背景的格子中，色调均为 0，即主色调为红色。饱和度和亮度的变化，都是基于红色色调而改变的颜色，这样搭配的颜色方案，是非常和谐的。当然，这里的颜色不够详细，可以调整饱和度和亮度，调出满意的颜色。

根据这个原理，我们可以对每种色调，进行同样的饱和度和亮度的调整，这样网页整体上不会很花哨。

提示：本节讲述的 HSL 色彩模式，很多人应该是第一次接触。也许在你的知识领域里遇到过 HSB 色彩模式，在 HSB 中，H（hues）表示色相，S（saturation）表示饱和度，B（brightness）表示发光亮度。但这是两种完全不同的色彩模式，HSB 色彩模式暂时不能用于 CSS，请勿混淆。

3.2.2　含不透明度的——HSLA 色彩模式

HSLA 色彩模式是 HSL 色彩模式的延伸，在色调、饱和度和亮度三个要素的基础上增加了不透明的参数。使用 HSLA 色彩模式，可以设计多种方式的透明效果。

1. 参数说明

HSLA 色彩模式的表示语法如下：

```
hsla(<length>, <percentage>, <percentage>,<alpha>)
```

参数及取值说明：前三个参数与 HSL 色彩模式的含义及用法完全相同。<alpha>表示不透明度，取值在 0 到 1 之间。取值为 1 时，与 HSL 色彩模式效果相同。

2. 示例介绍

使用 HSLA 色彩模式为背景的页面元素，显示为半透明效果。下面看一个示例。

【示例 3-11】 HSLA 半透明效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title> HSLA 半透明效果</title>
<style type="text/css">
ul {
    list-style:none;
    margin:10px;
    padding:0;
    background:url(../images/charactor.png) 10px 0 no-repeat;
}
li {height:20px;}
li:nth-child(1) {background:hsla(40, 50%, 50%, 0.1);}
li:nth-child(2) {background:hsla(40, 50%, 50%, 0.2);}
li:nth-child(3) {background:hsla(40, 50%, 50%, 0.3);}
li:nth-child(4) {background:hsla(40, 50%, 50%, 0.4);}
li:nth-child(5) {background:hsla(40, 50%, 50%, 0.5);}
li:nth-child(6) {background:hsla(40, 50%, 50%, 0.6);}
li:nth-child(7) {background:hsla(40, 50%, 50%, 0.7);}
li:nth-child(8) {background:hsla(40, 50%, 50%, 0.8);}
li:nth-child(9) {background:hsla(40, 50%, 50%, 0.9);}
li:nth-child(10) {background:hsla(40, 50%, 50%, 1);}
</style>
<body>
<ul>
  <li></li><li></li><li></li><li></li><li></li>
  <li></li><li></li><li></li><li></li><li></li>
</ul>
</body>
</html>
```

运行结果如图 3-19 所示。

代码分析：如示例 3-11 中，为了衬托透明度，外框 ul 元素设置了一个文字背景图，li 元素里的背景颜色的不透明度逐步增加。如图 3-19 所示，随着颜色的不透明度加深，背景图片显示越来越模糊。

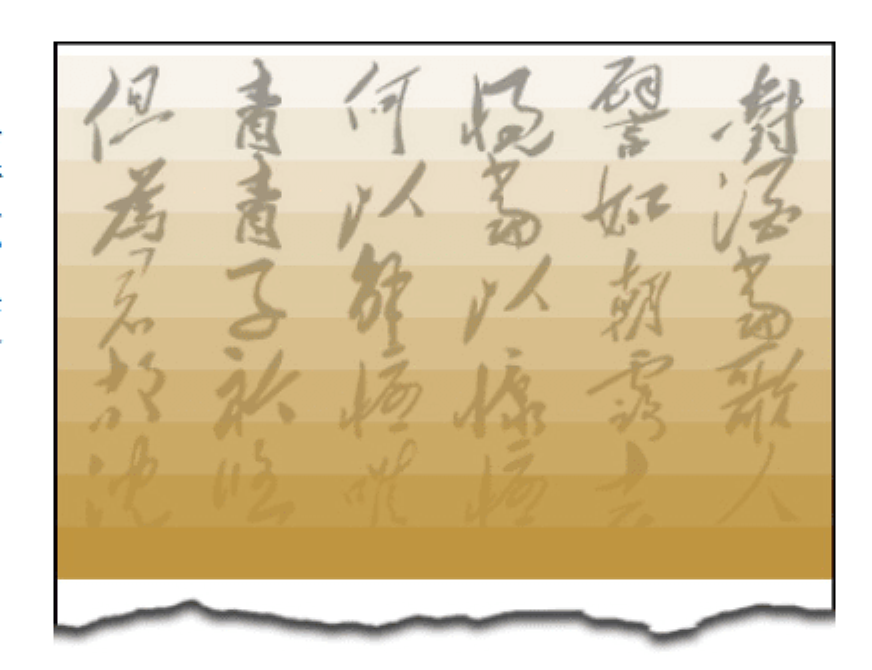


图 3-19　HSLA 半透明效果

3.2.3　含不透明度的——RGBA 色彩模式



RGBA 色彩模式是 RGB 色彩模式的延伸。在红、绿、

蓝、三原色的基础上增加了不透明度参数。使用 HSLA 色彩模式，也可以设计多种方式的透明效果。

1. 参数说明

RGBA 色彩模式的表示语法如下：

```
rgba(<red>, <green>, <blue>,<alpha>)
```

参数及取值说明如下。

前三个参数<red>、<green>、<blue>分别表示红色值、绿色值、蓝色值各自的取值。取值范围可以是正整数 0 到 255，也可以是百分数值范围 0.0%到 100.0%。但百分数值在有些浏览器中不支持，所以要慎用。<alpha>表示不透明度，取值在 0 到 1 之间。取值为 1 时，与 RGB 色彩模式效果相同。

2. 示例介绍

使用 RGBA 色彩模式为背景的页面元素，显示为半透明效果。与 HSLA 色彩模式用法很类似，直接改编示例 3-11 中的样式表如下。

【示例 3-12】 RGBA 半透明效果。

```
<style type="text/css">
ul {
    list-style:none;
    margin:10px;
    padding:0;
    background:url(../images/charactor.png) 10px 0 no-repeat;
}
li {height:20px;}
li:nth-child(1) {background:rgba(255, 153, 0, 0.1);}
li:nth-child(2) {background:rgba(255, 153, 0, 0.2);}
li:nth-child(3) {background:rgba(255, 153, 0, 0.3);}
li:nth-child(4) {background:rgba(255, 153, 0, 0.4);}
li:nth-child(5) {background:rgba(255, 153, 0, 0.5);}
li:nth-child(6) {background:rgba(255, 153, 0, 0.6);}
li:nth-child(7) {background:rgba(255, 153, 0, 0.7);}
li:nth-child(8) {background:rgba(255, 153, 0, 0.8);}
li:nth-child(9) {background:rgba(255, 153, 0, 0.9);}
li:nth-child(10) {background:rgba(255, 153, 0, 1);}
</style>
```

运行结果如图 3-20 所示。

代码分析：在示例 3-12 中，为了衬托透明度，外框 ul 元素也设置了一个文字背景图，li 元素里的背景颜色的不透明度逐步增加。如图 3-20 所示，随着颜色的不透明度加深，背景图片显示越来越模糊。

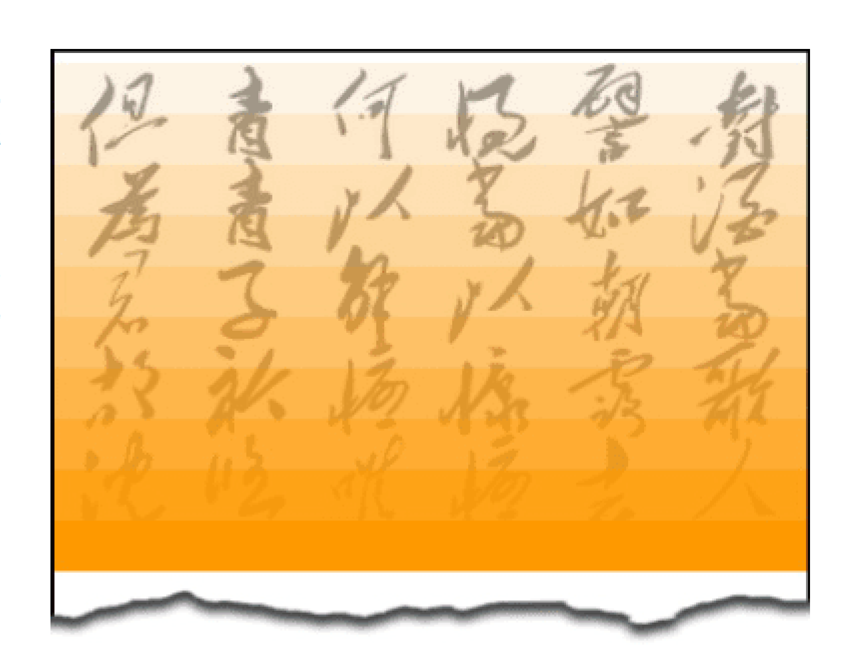


图 3-20　RGBA 半透明效果

3.2.4　不透明度——opacity 属性

在 CSS 3 中，除了 HSLA 和 RGBA 两种色彩模式可以设置半透明效果之外，还有专门的不透明属性 opacity，可

以设置半透明效果。该属性可以应用于任何页面元素中。

1. 参数说明

opacity 属性的语法如下：

```
opacity : <alpha> | inherit
```

参数及取值说明如下。

- <alpha>：表示不透明度，取值在 0 到 1 之间。默认为 1，表示完全不透明；0 表示完全透明。
- inherit：该值表示继承父元素的不透明度。

提示： 在 IE 8 及以前的浏览器版本中的透明效果，使用 filter 来设置：filter:alpha(opacity=<value>)，<value>的取值范围与 opacity 属性的<alpha>相同。

2. 示例介绍

使用 opacity 属性，可针对某个页面元素设置半透明效果。

【示例 3-13】 图片的半透明效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>半透明效果</title>
<style type="text/css">
ul {
    list-style:none;
    margin:10px;
    padding:0;
    background:url(../images/charactor.png);
    height:160px;
}
li {
    float:left;
    margin:5px;
    width:200px;
    height:150px;
}
li:nth-child(1) {opacity:0.5;}
li:nth-child(2) {opacity:0.8; }
</style>
<body>
<ul>
  <li><img src="../images/bird.png" alt="鸟" /></li>
  <li><img src="../images/bird.png" alt="鸟" /></li>
  <li><img src="../images/bird.png" alt="鸟" /></li>
</ul>
</body>
</html>
```

运行结果如图 3-21 所示。

代码分析：在示例 3-13 中，为了展示半透明效果，在 ul 元素中设置了文字背景图。我们为三个 li 元素内都添加了图片，并针对 li 元素分别设置了不同的 opacity 属性值。由

图 3-21 所示，li 内部的图片内容也应用了半透明效果。



图 3-21　页面的半透明效果

提示：opacity 属性的不透明度，可以应用于各个页面元素及其内部的内容。

3.2.5　实验室：半透明的遮蔽层

半透明的遮蔽层是网页中常用的表现形式。常常为了突出弹出层的内容，需要一个半透明的遮蔽层来遮挡页面的其他内容，以增强用户体验。下面就用学过的不透明度知识来实现图片的预览。

【示例 3-14】 半透明的遮蔽层。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>半透明的遮蔽层</title>
<style type="text/css">
ul {
    list-style:none;
    margin:10px;
    padding:0;
    height:160px;
}
li {
    float:left;
    margin:5px;
    width:100px;
    height:80px;
}
li img {
    width:100px;
    height:80px;
    opacity:0.5;                          /* 列表图片的不透明度设置为 0.5 */
    cursor:pointer;
}
li img:hover {
    opacity:1;                            /* 列表图片的不透明度设置为 1 */
}
.bg {
    position:absolute;
    background-color:hsla(0, 0%, 50%, 0.5);
```

```
                                /* 遮蔽层的背景设置为不透明度为 0.5 的黑色 */
    top:0;
    left:0;
}
.box {
    position:absolute;
    top:130px;
    left:150px;
    z-index:99;
    border-radius:4px;
    padding:5px;
    background-color:#fff;
    line-height:20px;
    font-size:12px;
    color:#666;
    font-weight:bold;
}
.box a {
    display:block;
    position:absolute;
    z-index:100;
    top:-8px;
    left:498px;
    border-radius:9px;
    border:2px solid #e4e4e4;
    background-color:#bbb;
    line-height:14px;
    width:14px;
    text-align:center;
    font-family:Arial, Helvetica, sans-serif;
    font-size:14px;
    color:#FFF;
    text-decoration:none;
}
.box img {
    width:300px;
    height:200px;
}
</style>
<script type="text/javascript">
        //打开图片预览
    function popbox(obj){
        //创建半透明的遮蔽层
        document.body.style.overflow="hidden";
        var w = window.innerWidth;
        var h = window.innerHeight;
        var bgdiv = document.createElement("div");
        bgdiv.style.width = w + "px";
        bgdiv.style.height = h + "px";
        bgdiv.className = "bg";
        bgdiv.id = "bg";
        document.body.appendChild(bgdiv);
        //创建图片预览层
        var box = document.createElement("div");
        box.id = "floatbox";
        box.className = "box";
        box.innerHTML="<a href='javascript:closebox()'>&times;</a>";
        box.innerHTML+="<img src="+ obj.src+" />";
        if(obj.alt){
            box.innerHTML+="<div>"+ obj.alt+"</div>";
```

```
        }
        //图片预览的居中定位
        box.style.left=(w-510)/2+"px";
        box.style.top=(h-405)/2+"px";
        document.body.appendChild(box);
    }
        //关闭图片预览
    function closebox(){
        document.body.style.overflow="";
        var bg=document.getElementById("bg");
        bg.parentNode.removeChild(bg);
        var box=document.getElementById("floatbox");
        box.parentNode.removeChild(box);
    }
</script>
<body>
<ul>
  <li><img src="../images/J10.jpg" alt="歼 10 战斗机" onclick="popbox(this)"
  /></li>
  <li><img src="../images/F22.jpg" alt="F22 战斗机" onclick="popbox(this)"
  /></li>
  <li><img src="../images/Su35.jpg" alt="苏 35 战斗机" onclick="popbox
  (this)" /></li>
</ul>
</body>
</html>
```

运行结果如图 3-22 所示。



图 3-22　半透明遮蔽层

代码分析：在示例 3-14 中，设置了列表图片的不透明度 0.5，鼠标滑过图片，不透明度变为 1，即图片变得清晰；把遮蔽层的 CSS 类样式（.bg）的背景颜色指定为带不透明度设置的颜色，来实现半透明效果，该类样式会在图片函数 popbox()中调用。

3.3　背　　景

在布局和美化网页方面，常常离不开对背景的设置。在传统的 CSS 背景设计中，由于功能的局限，设计师的灵感难以尽情发挥。为了使背景设计更加灵活，CSS 3 增强了原有背景属性的功能，并增添了一些新的背景属性。不但可以在同一元素内叠加多个背景图像，也可以对背景图像的原点位置、显示区域和大小方面进行调整和控制。

3.3.1　元素里定义多个背景图片

CSS 3 中，可以对一个元素应用一个或多个图片作为背景。代码和 CSS 2 中的一样，只需要用逗号来区别各个图片。第一个声明的图片定位在元素顶部，其他的图片依次在其下排列。

1. 参数说明

定义页面元素的背景语法如下：

```
Background : [background-image] | [background-origin] | [background-clip]
| [background-repeat] | [background-size] | [background-position]
```

参数及取值说明如下。

- <background-image>：指定或检索对象的背景图像。
- <background-origin>：指定背景的原点位置，属于新增的属性。
- <background-clip>：指定背景的显示区域，属于新增的属性。
- <background-repeat>：设置或检索对象的背景图像是否及如何重复铺排。
- <background-size>：指定背景图片的大小，属于新增的属性。
- <background-position>：设置或检索对象的背景图像位置。

如果定义多重背景图，则用逗号隔开各个背景图设置。如果使用子属性直接定义，那么各个子属性也用逗号对应依次隔开。

2. 示例介绍

下面就通过一个示例来展示多重背景的定义方法。

【示例 3-15】 多重背景效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>多重背景</title>
<style type="text/css">
body{
    background:url(../images/icon12.png) 120px 110px no-repeat,
            url(../images/icon5.png) 400px 10px no-repeat,
            url(../images/J10-2.jpg) no-repeat;
}
```

```
</style>
<body>
</body>
</html>
```

运行结果如图 3-23 所示。

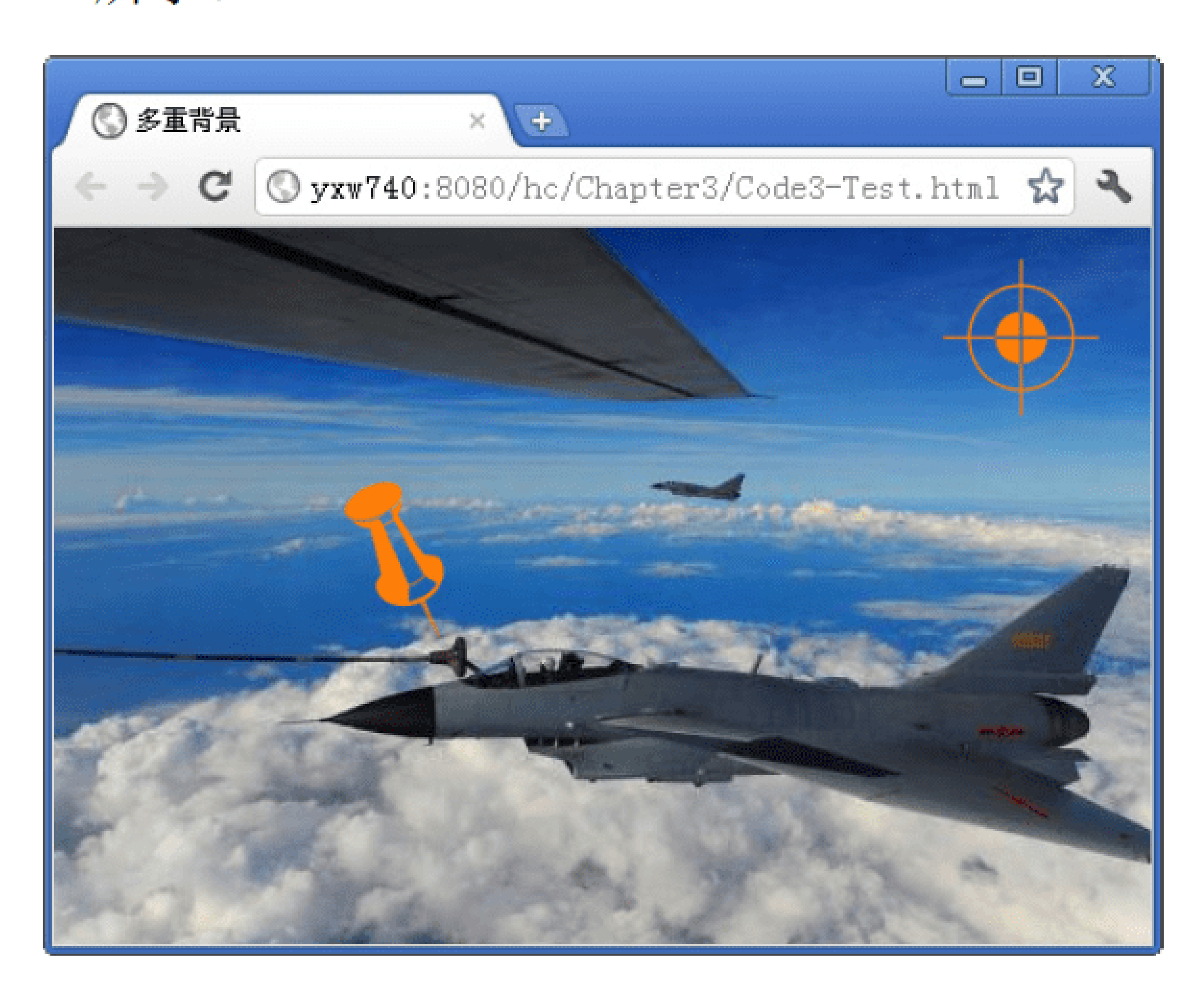


图 3-23　多重背景合成效果

代码分析：在示例 3-15 中，设置了三个图片背景，中间用逗号隔开，均不重复铺排。写在前面的背景图像会显示在上面，写在后面的背景图像则显示在下面。

由于 background 属性是由众多子属性组成的，因此示例 3-15 中的样式也可以写为：

```
body{
    background-image : url(../images/icon12.png) , url(../images/
    icon5.png), url(../images/J10-2.jpg);
    background-position : 120px 110px , 400px 10px , 0 0;
    background-repeat : no-repeat , no-repeat , no-repeat;
}
```

运行后有同样的结果，如图 3-23 所示。

3.3.2　指定背景的原点位置

CSS 3 的新增属性 background-origin 用来指定背景图像的原点位置。在默认情况下，属性 background-position 总是以元素边框以内的左上角为坐标原点进行背景图像定位。使用 background-origin 属性可以对该原点位置进行调整。

1. 参数说明

属性 background-origin 的语法如下：

```
background-origin: border-box | padding-box | content-box
```

取值说明如下。

- ❑ border-box：原点位置为边框（border）区域的开始位置。
- ❑ padding-box：原点位置为内边距（padding）区域的开始位置。

- content-box：原点位置为内容（content）区域的开始位置。

可见，该原点位置不是通过直接给出原点坐标指定的，而是根据盒模型的结构来确定的，这对于网页背景的布局有一定的优势。关于盒模型结构如图 3-24 所示。

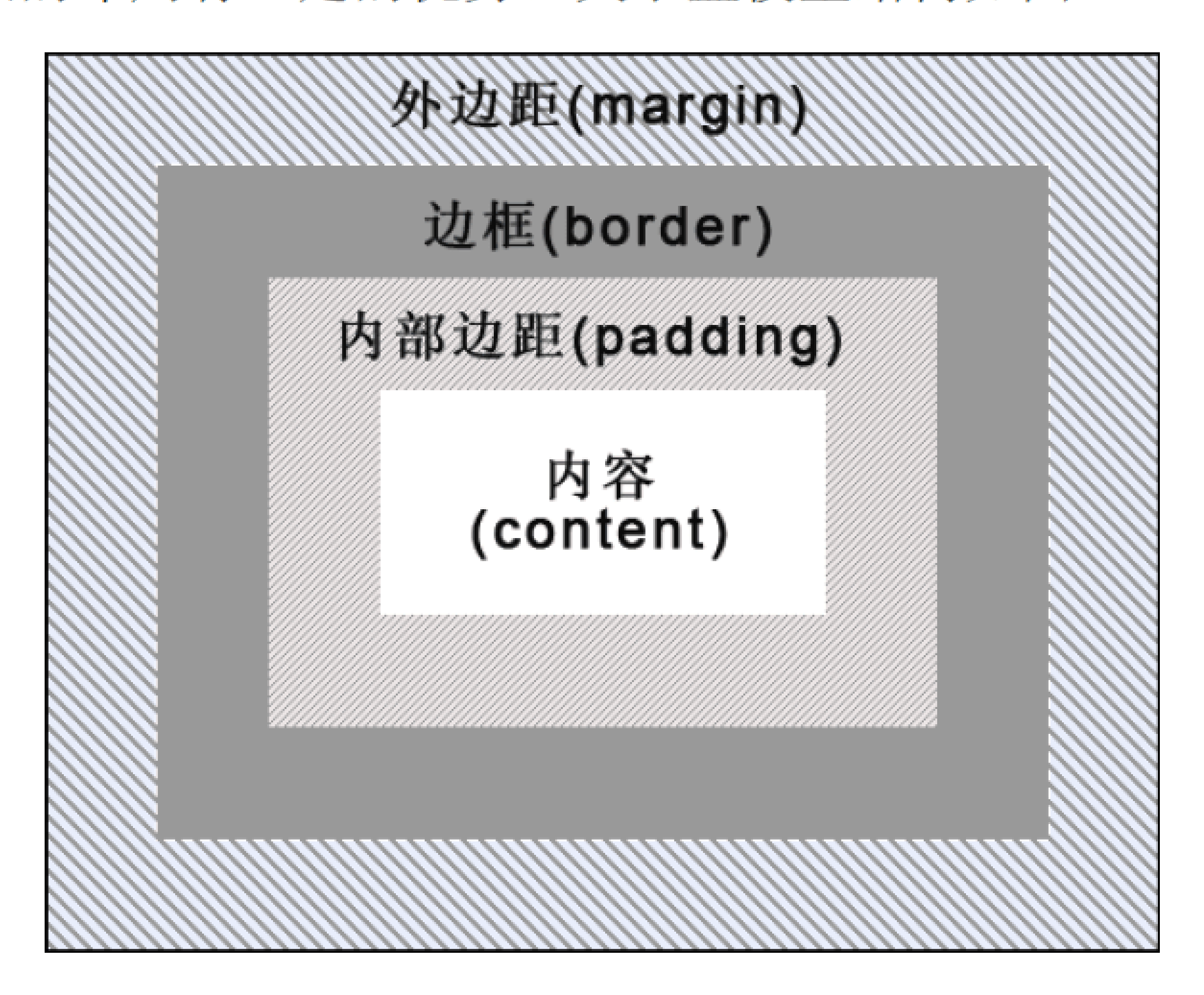


图 3-24　盒模型结构图

提示： 在之前的部分浏览器中，background-origin 属性的取值可以为：border、padding 和 content，但是不建议使用，因为不符合最新的 CSS 3 规范，而且主流浏览器对符合规范的取值的支持更加良好。

2. 示例介绍

下面通过示例来学习 background-origin 属性各个值的应用效果。

【示例 3-16】 背景的原点位置。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>背景的原点位置</title>
<style type="text/css">
div {
    padding:50px;                                    /* 设置内边距为 50px */
    border:50px solid rgba(255, 153, 0, 0.6);   /* 设置边框宽度为 50px */
    height:100px;
    width:200px;
    color:#fff;
    font-size:24px;
    font-weight:bold;
    text-shadow:2px 0 1px #f00,-2px 0 1px #f00,
            0 2px 1px #f00, 0 -2px 1px #f00;
    background-image:url(../images/Bridge.jpg);    /* 设置背景图像 */
    background-position:0 0;                       /* 背景图像起始位为原点 */
    background-repeat:no-repeat;                   /* 背景图像不平铺 */
    -webkit-background-origin:border-box;  /* 原点为边框的开始（webkit） */
    -moz-background-origin:border-box;     /* 原点为边框的开始（moz） */
```

```
    background-origin:border-box;             /* 原点为边框的开始 */
}
</style>
<body>
<div>内容从这里开始</div>
</body>
</html>
```

运行结果如图 3-25 所示。



图 3-25　原点为边框的开始

代码分析：在示例 3-16 中，设置了背景的起始点为原点，背景开始的位置即是原点位置；为了表现原点从边框开始，这里设置了宽度为 50px 的边框，且边框颜色为半透明，便于看见边框下的背景。对于属性 background-origin，由于兼容性问题，需要针对多种浏览器内核分别写一个；为了突出内容边界的位置，该示例也添加了文字样式。如图 3-25 所示，背景图片是从边框内开始的，原点位置即是边框左上角位置。

下面我们将属性 background-origin 分别修改为 padding-box 或 content-box，如：

```
-webkit-background-origin:padding-box;        /* 原点为内边距的开始（webkit） */
-moz-background-origin: padding-box;          /* 原点为内边距的开始（moz） */
background-origin: padding-box;               /* 原点为内边距的开始 */
```

或

```
-webkit-background-origin: content-box;       /* 原点为内容的开始（webkit） */
-moz-background-origin: content-box;          /* 原点为内容的开始（moz） */
background-origin: content-box;               /* 原点为内容的开始 */
```

运行结果分别如图 3-26、图 3-27 所示。



图 3-26　原点为内边距的开始

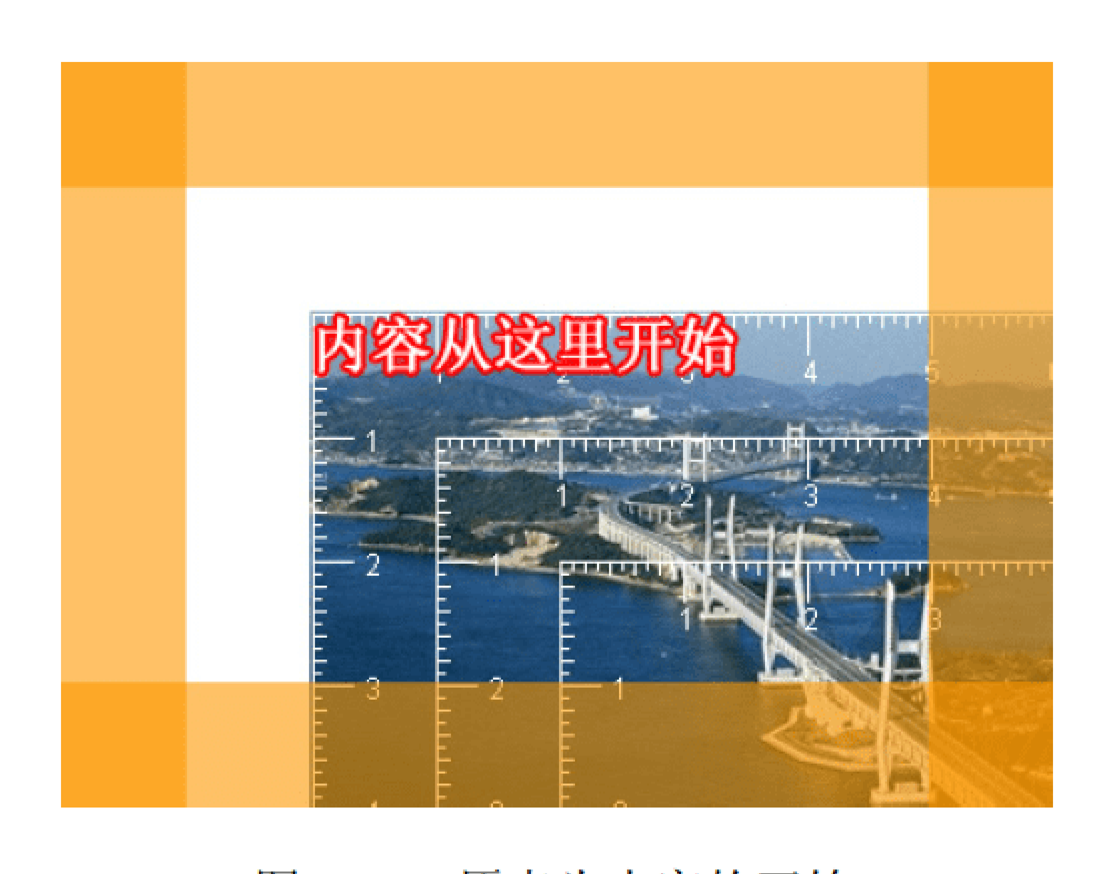


图 3-27　原点为内容的开始

说明： 默认情况下，背景从边框开始显示。在调整属性 background-origin 的值时，背景图像的左上角位置发生了变化，如图 3-26 和图 3-27 所示，右下方的背景仍然会显示在边框区域或内边距区域。

3.3.3　指定背景的显示区域

CSS 3 的新增属性 background-clip，用来指定背景的显示区域。在支持 CSS 3 的环境下，背景的显示区域是包含元素边框在内的，实际使用中，或许仅在内容区域显示背景。在 CSS 3 中，可以使用属性 background-clip 来修改显示区域。

1. 参数说明

属性 background-clip 的语法如下：

```
background-clip: border-box | padding-box | content-box
```

取值说明如下。

- border-box：背景从边框（border）开始显示。
- padding-box：背景从内边距（padding）开始显示。
- content-box：背景仅在内容（content）区域显示。

可见，该属性的使用方法与属性 background-origin 一样，其值也是根据盒模型的结构来确定的。这两个属性常常会结合起来使用，以达到对背景的灵活控制。

提示： 在之前的部分浏览器中，background-clip 属性的取值可以为：border、padding 和 content，但是不建议使用，因为不符合最新的 CSS 3 规范，而且主流浏览器对符合规范的取值的支持更加良好。

2. 示例介绍

下面通过示例来对比学习 background-clip 属性的各个值的应用效果。

【示例 3-17】 背景的显示区域。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>背景的显示区域</title>
<style type="text/css">
div {
    padding:50px;                                    /* 设置内边距为 50px */
    border:50px solid rgba(255, 153, 0, 0.6);        /* 设置边框宽度为 50px */
    height:100px;
    width:200px;
    color:#fff;
    font-size:24px;
    font-weight:bold;
    text-shadow:2px 0 1px #f00,
            -2px 0 1px #f00,
            0 2px 1px #f00,
            0 -2px 1px #f00;
    background-image:url(../images/Bridge.jpg);      /* 设置背景图像 */
    background-position:0 0;                         /* 背景图像起始位为原点 */
    background-repeat:no-repeat;                     /* 背景图像不平铺 */
```

```
    -webkit-background-origin:border-box;   /* 原点从边框开始（webkit） */
    -moz-background-origin:border-box;      /* 原点从边框开始（moz） */
    background-origin:border-box;           /* 原点从边框开始 */

    -webkit-background-clip:border-box;     /* 背景从边框开始显示（webkit） */
    -moz-background-clip:border-box;        /* 背景从边框开始显示（moz） */
    background-clip:border-box;             /* 背景从边框开始显示 */
}
</style>
<body>
<div>内容从这里开始</div>
</body>
</html>
```



图 3-28　背景从边框开始显示

运行结果如图 3-28 所示。

代码分析：示例 3-17 与示例 3-16 基本类似。在示例 3-17 中，设置了背景的起始点为原点，背景开始的原点位置为边框的开始位置；同样设置了半透明边框便于观察；内容区域也添加了文字。如图 3-28 所示，背景从边框（border）内开始显示。

用同样的方法，我们将属性 background-clip 分别修改为 padding-box 或 content-box，如：

```
-webkit-background-clip:padding-box;   /* 背景从内边距开始显示（webkit） */
-moz-background-clip: padding-box;     /* 背景从内边距开始显示（moz） */
background-clip: padding-box;          /* 背景从内边距开始显示 */
```

或

```
-webkit-background-clip: content-box;  /* 背景仅在内容区域显示（webkit） */
-moz-background-clip: content-box;     /* 背景仅在内容区域显示（moz） */
background-clip: content-box;          /* 背景仅在内容区域显示 */
```

运行结果分别如图 3-29、图 3-30 所示。



图 3-29　背景从内边距开始显示



图 3-30　背景仅在内容区域显示

说明：图 3-29 和图 3-30 所示，由于背景显示区域的限制，背景图像被裁剪了，所以该显示区域也叫裁剪区域，该属性 background-clip 也叫背景裁剪属性。该示例也使

用了 background-origin 属性，以设置图像的起始位置。说明这两个属性常常会一起使用。

3.3.4　指定背景图像的大小

CSS 3 的新增属性 background-size，用来指定背景图像的大小。在传统的 CSS 设计中，背景图像的大小是不可以改变的，通常会把同样的图片，做成不同的尺寸，以适应不同大小的背景显示。在 CSS 3 中，通过 background-size 属性，可以有多种方式来指定背景图像的大小，以适应不同情况下的需要。

1. 参数说明

属性 background-size 的语法如下：

```
background-size : [ <length> | <percentage> | auto ]{1,2} | cover | contain
```

取值说明如下。

- <length>：由浮点数字和单位标识符组成的长度值，不可为负值。
- <percentage>：取值为 0%到 100%之间的值，是基于背景图像的父元素的百分比。
- cover：保持背景图像本身的宽高比例，将图像缩放到正好完全覆盖所定义的背景区域。
- contain：保持背景图像本身的宽高比例，将图像缩放到正好适应所定义的背景区域。

background-size 属性可以使用<length>或<percentage>来设置背景图片的高度和宽度。第一个值设置宽度，第二个值设置高度；如果只给出一个值，第二个值设置为“auto”。

需要注意的是，<length>是直接指定背景图像的宽和高；<percentage>是基于背景图像的父元素尺寸的百分比，来确定背景图像的宽和高，其中父元素的计算尺寸包含父元素的内边距，不包括边框。

2. 示例介绍

background-size 属性的使用方法比较灵活，下面通过示例来说明。

【示例 3-18】 不同尺寸的多重背景图像。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>不同尺寸的多重背景图像</title>
<style type="text/css">
div {
    margin:10px;
    padding:50px;
    border:1px solid #000;
    height:220px;
    width:400px;
    background-repeat:no-repeat;
    background-image:url(../images/WestLake.jpg)   /* 最前面背景图像 */
```

```
                ,url(../images/WestLake.jpg)   /* 中间背景图像 */
                ,url(../images/WestLake.jpg);  /* 最后面背景图像 */
    background-size:30% 30%                    /* 最前面背景图像尺寸 */
                ,60% 60%                       /* 中间背景图像尺寸 */
                ,100% 100%;                    /* 最后面背景图像尺寸 */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-31 所示。



图 3-31　不同尺寸的多重背景图像

代码分析：在示例 3-18 中，设置三个图片相同的背景图像，然后分别设置尺寸。这里是使用百分比数值来设定背景图像大小的。如图 3-31 所示，背景图像会根据设置的图像大小分别显示。

3. 值cover与值contain

在前面介绍 background-size 语法的时候，介绍了这两个值，它们的意思非常相近。为避免混淆，这里通过一个示例来比较它们之间的异同。

【示例 3-19】 值 cover 与值 contain。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>值 cover 与值 contain</title>
<style type="text/css">
div {
    margin:10px;
    padding:50px;
    border:1px solid #000;
    height:150px;
    width:400px;
    background-repeat:no-repeat;
```

```
    background-image:url(../images/WestLake.jpg);      /* 背景图像 */
    background-size:cover;                             /* 值 cover */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-32 所示。

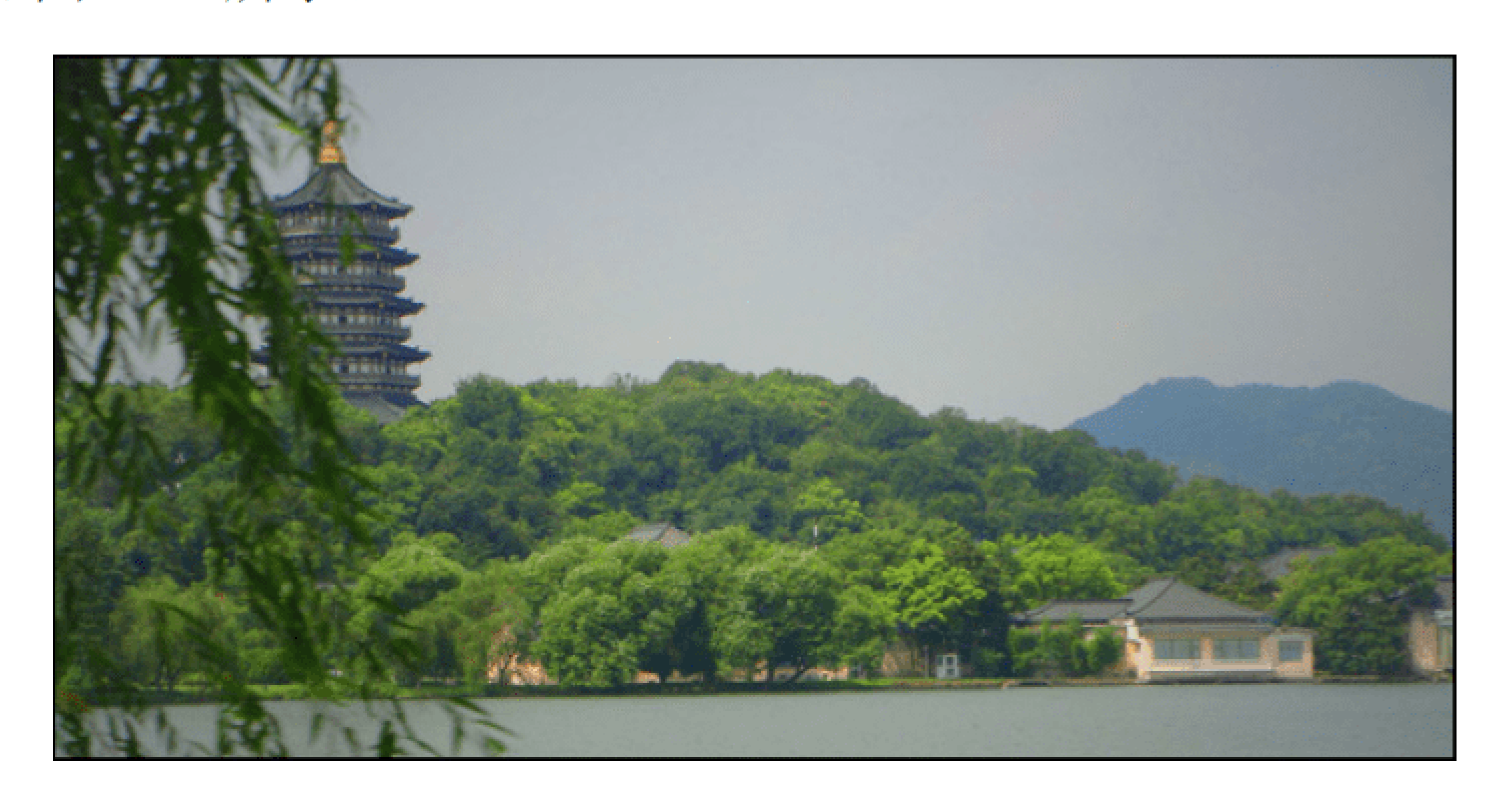


图 3-32　图像覆盖整个背景区域

如果将属性 background-size 的值设置为 contain，更改如下：

```
background-size: contain;                        /* 值 contain*/
```

其运行结果如图 3-33 所示。



图 3-33　图像未覆盖整个背景区域

代码分析：在示例 3-19 中，背景图像都产生了缩放。不同的是，当属性 background-size 的值为 cover 时，背景图像按比例缩放，直至覆盖整个背景区域为止，但可能会裁剪掉部分图像，如图 3-32 所示。当属性 background-size 的值为 contain 时，背景图像会完全显示出来，但可能不会完全覆盖背景区域，如图 3-33 所示。

3.3.5　实验室：设计信纸的效果

本节我们就用学过的背景样式知识，制作一个信纸的效果。仍然使用多重背景的方法，

把多个图片同时作为背景来显示，以实现信纸效果。

1. 背景图像素材

本节所介绍的案例中，将会使用 6 个图片作为背景图像素材。其中信纸的 4 个角各有一个背景图片，信纸本身的沙砾效果的背景和横线背景，如图 3-34 所示。

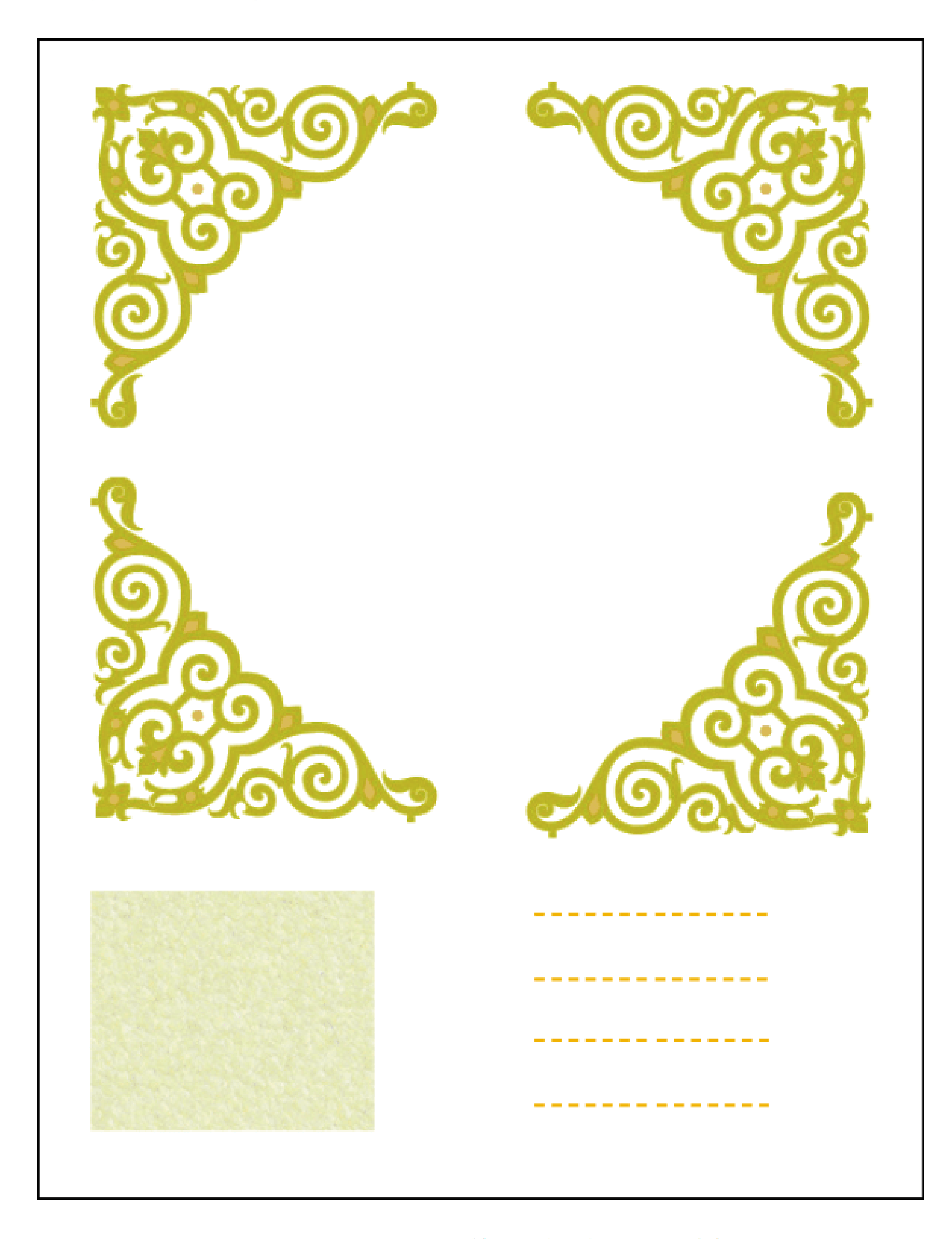


图 3-34　设计信纸的背景素材

【示例 3-20】 信纸的效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>信纸的效果</title>
<style type="text/css">
div {
    padding:25px;
    border:10px solid rgba(204, 204, 51, 0.8);
    height:300px;
    width:220px;
    font-size:12px;
    line-height:22px;
    background-image:url(../images/line.png),        /* 第一个背景：横线 */
                url(../images/left-top.png),         /* 第二个背景:左上角背景 */
                url(../images/right-top.png),        /* 第三个背景:右上角背景 */
                url(../images/left-bottom.png),      /* 第四个背景:左下角背景 */
                url(../images/right-bottom.png),     /* 第五个背景:右下角背景 */
                url(../images/wenli.jpg);            /* 第六个背景:纹理背景 */
```

```
    background-repeat:repeat,          /* 第一个背景：平铺 */
                no-repeat,             /* 第二个背景：不平铺*/
                no-repeat,             /* 第三个背景：不平铺*/
                no-repeat,             /* 第四个背景：不平铺*/
                no-repeat,             /* 第五个背景：不平铺*/
                repeat;                /* 第六个背景：平铺*/

    background-position:left top,      /* 第一个背景：左上*/
                left top,              /* 第二个背景：左上*/
                right top,             /* 第三个背景：右上*/
                left bottom,           /* 第四个背景：左下*/
                right bottom,          /* 第五个背景：右下*/
                left top;              /* 第六个背景：左上*/

    background-size: 40px 44px,        /* 第一个背景：固定尺寸*/
                20%,                   /* 第二个背景：宽为父元素 20%的大小，高自动*/
                20%,                   /* 第三个背景：宽为父元素 20%的大小，高自动*/
                20%,                   /* 第四个背景：宽为父元素 20%的大小，高自动*/
                20                     /* 第五个背景：宽为父元素 20%的大小，高自动*/
                20%;                   /* 第六个背景：宽为父元素 20%的大小，高自动*/

    background-origin: content-box,    /* 第一个背景：原点为内容区域开始 */
                border-box,            /* 第二个背景：原点为边框区域开始 */
                border-box,            /* 第三个背景：原点为边框区域开始 */
                border-box,            /* 第四个背景：原点为边框区域开始 */
                border-box,            /* 第五个背景：原点为边框区域开始 */
                padding-box;           /* 第六个背景：原点为内边距区域开始 */

    background-clip: content-box,      /* 第一个背景：背景仅在内容区域显示 */
                border-box,            /* 第二个背景：背景从边框区域开始显示 */
                border-box,            /* 第三个背景：背景从边框区域开始显示*/
                border-box,            /* 第四个背景：背景从边框区域开始显示*/
                border-box,            /* 第五个背景：背景从边框区域开始显示*/
                padding-box;           /* 第六个背景：背景从内边距区域开始显示*/
}
div h1 {
    margin:0;
    padding:0;
    font-size:14px;
}
div p {
    margin:0;
    padding:0;
    text-indent:2em;
}
</style>
<body>
<div>
  <h1>致加西亚的一封信</h1>
  <p>在所有与古巴有关的事情中，有一个人常常令我无法忘怀。</p>
  <p>美西战争爆发以后，美国必须马上与西班牙反抗军首领加西亚将军取得联系。加西亚将军隐藏在古巴辽阔的崇山峻岭中——没有人知道确切的地点，因而无法送信给他。但是，美国总统必须尽快地与他建立合作关系。怎么办呢？</p>
```

```
  <p>有人对总统推荐说：&ldquo;有一个名叫罗文的人，如果有人能找到加西亚将军，那个人一定就是他。&rdquo;  </p>
  <p>......</p>
</div>
</body>
</html>
```

运行结果如图 3-35 所示。

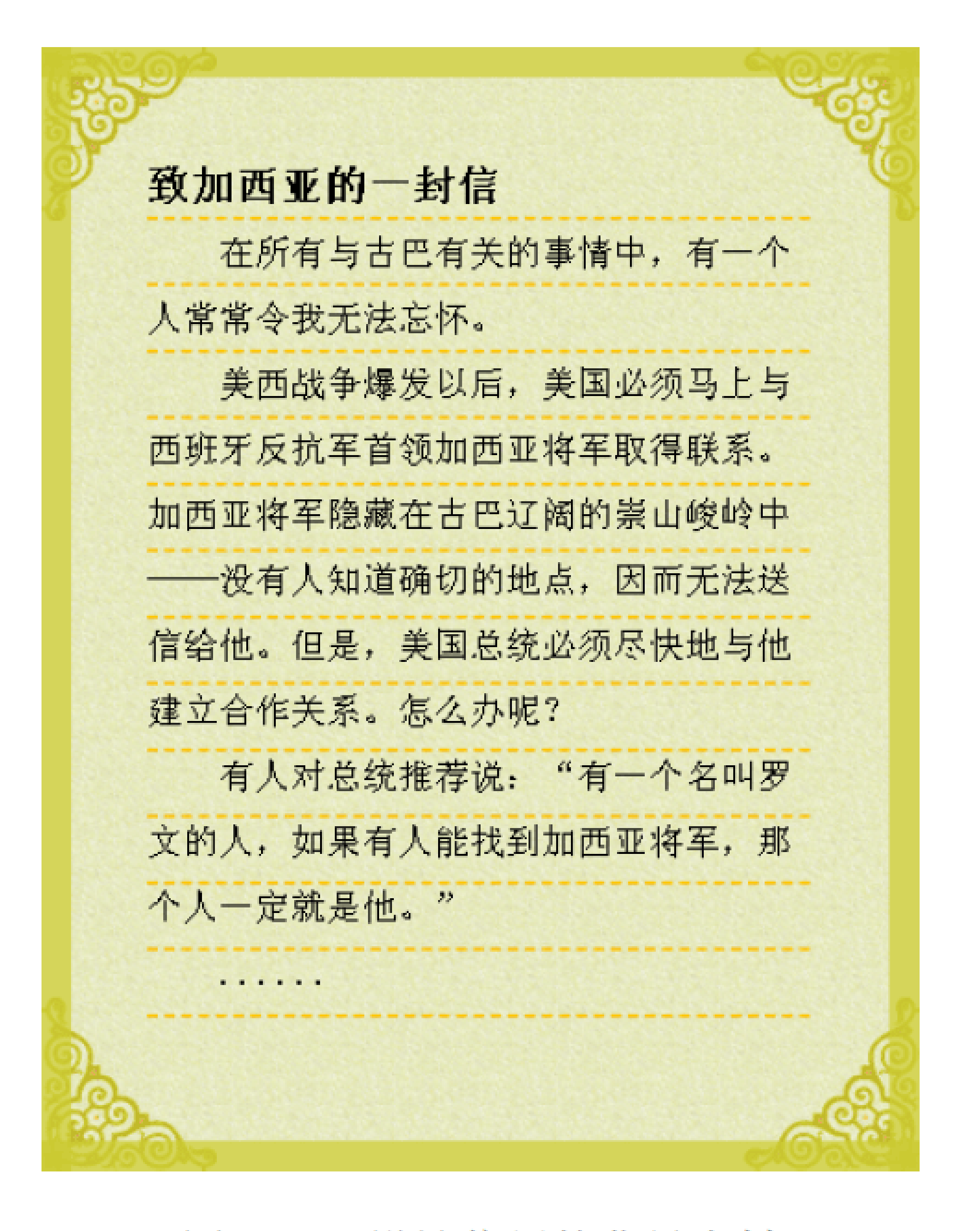


图 3-35　设计信纸的背景素材

代码分析：在示例 3-20 中，同时设置了 6 个背景图片。为了能灵活地控制这些背景图片，对每一个属性都用逗号间隔成 6 个部分，以达到分别控制各个背景图像的目的。在该示例中，同时使用了属性 background-size、属性 background-origin 和属性 background-clip，足见新增属性的强大与便利。

3.4　边　　框

在网页设计中，边框是常用的美化手法之一。在 CSS 3 之前，页面边框比较单调，只能设置边框的粗细程度和边框颜色，如果想使边框的效果更加丰富，只能事先设计好边框图片，然后通过使用背景或直接插入图片的方式来实现。在 CSS 3 中，通过样式表设置，可以直接实现诸如圆角边框、图像边框和多色边框等效果。这对前端开发人员来说，无疑是一件可喜的事情。

3.4.1　设计圆角边框——border-radius 属性

边框在网页里几乎随处可见，同时也显得边框非常重要。border-radius 属性为 CSS 3

的新增属性，用来设计边框的圆角，相信前端开发人员已经期待很久了吧。

1. 参数说明

border-radius 属性的语法如下：

```
border-radius : none | <length>{1,4} [ / <length>{1,4} ]?
```

取值说明如下。

- none：默认值，表示元素没有圆角。
- <length>：由浮点数字和单位标识符组成的长度值，不可为负值。该值分两组，每组可以有 1 到 4 个值。第一组为水平半径，第二组为垂直半径，如果第二组省略，则默认等于第一组的值。

2. 派生子属性

border-radius 属性，又针对边框的 4 个角，派生出 4 个子属性，如下所示。

- border-top-left-radius：定义左上角的圆角。
- border-top-right-radius：定义右上角的圆角。
- border-bottom-left-radius：定义左下角的圆角。
- border-bottom-right-radius：定义右下角的圆角。

如果边框的 4 个圆角的半径各不相同，使用子属性单独定义每个圆角，是一个直接有效的方法。

3. 圆角边框示例

border-radius 属性本身又包含 4 个子属性，当为该属性赋一组值的时候，将遵循 CSS 的赋值规则。从 border-radius 属性语法可以看出，其值也可以同时包含 2 个值、3 个值或 4 个值，多个值的情况用空格间隔开。

下面看一个示例。该示例中只有一个 div 元素，使用 border-radius 属性把边框美化成圆角。

【示例 3-21】 设计圆角边框。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>边框圆角属性 border-radius</title>
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:# fe0;
    border: 10px solid #f90;        /* 宽度为 10px 的边框 */
    border-radius:20px;             /* 边框的圆角半径为 20px */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-36 所示。



图 3-36　圆角边框

代码分析：在示例 3-21 中，设置了一个宽度为 10px 的边框。仅仅添加了 border-radius 属性，并赋值为 20px，就把边框的 4 个角变成半径为 20px 的圆角。边框的圆角参照图 3-37 所示，20px 的圆角半径为边框的外边半径，内边半径仅为 10px。

如果边框的宽度大于或等于圆角半径，则边框内部将不再是圆角，如图 3-38 所示。

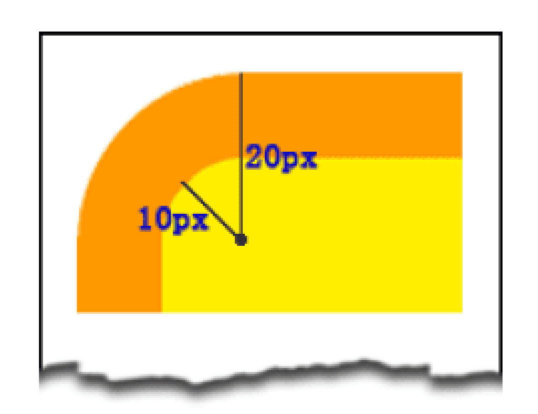




图 3-37　边框的宽度小于圆角半径

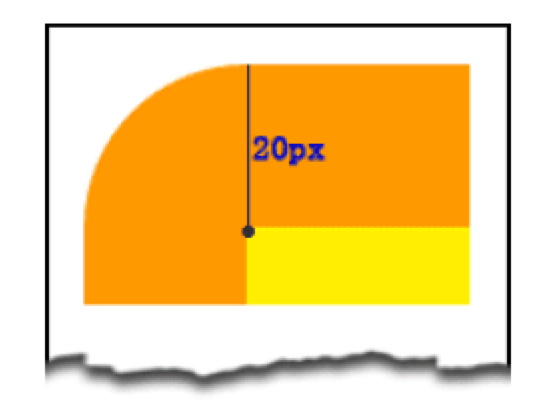


图 3-38　边框的宽度大于或等于圆角半径

虽然 border-radius 属性是与边框有关的属性，但是在不设置边框的情况下，如果该块元素有背景，则把背景的 4 个角定义为圆角。去除样式表中的边框样式如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:# fe0;
    border-radius:20px;              /* 边框的圆角半径为 20px */
}
</style>
```

运行结果如图 3-39 所示。



图 3-39　没有边框的圆角背景

遵循 CSS 的赋值规则，border-radius 属性可以被赋值为 2 个值的集合参数，则第一个值表示左上角和右下角的圆角半径，第二个值表示右上角和左下角的圆角半径。例如，调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
```

```
    background-color:# fe0;
    border: 10px solid #f90;        /* 宽度为 10px 的边框 */
    border-radius:20px 40px;        /* 边框的圆角半径为 20px */
}
</style>
```

运行结果如图 3-40 所示。

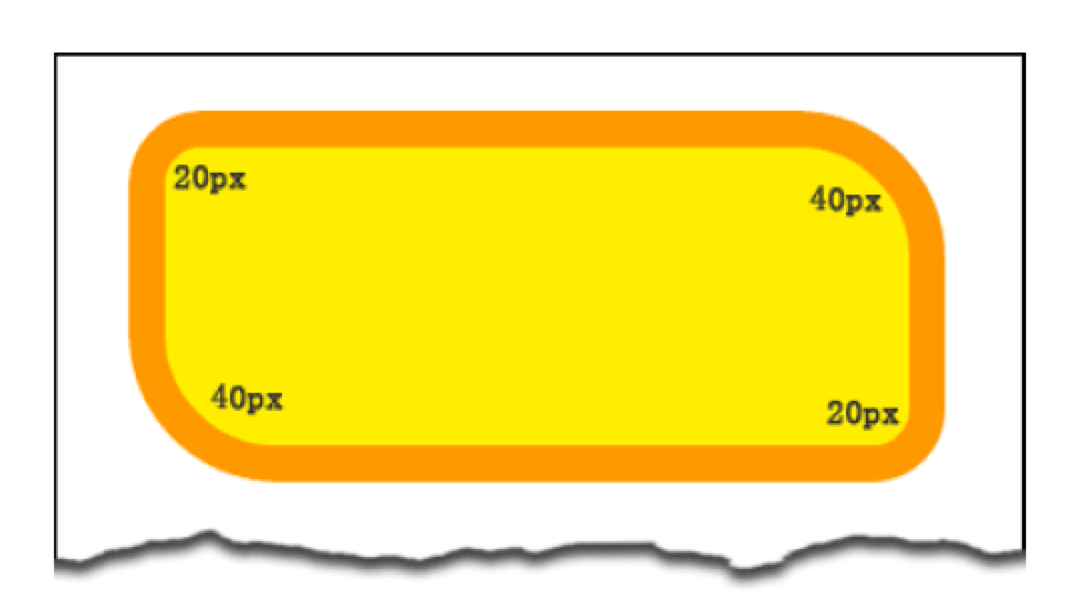


图 3-40　2 个参数的赋值

同样，把 border-radius 属性赋值为 3 个值的集合参数，则第一个值表示左上角的圆角半径，第二个值表示右上角和左下角的圆角半径，第三个值表示右下角的圆角半径。例如，调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:# fe0;
    border: 10px solid #f90;              /* 宽度为 10px 的边框 */
    border-radius:20px 40px 30px;         /* 边框的圆角半径为 20px */
}
</style>
```

运行结果如图 3-41 所示。

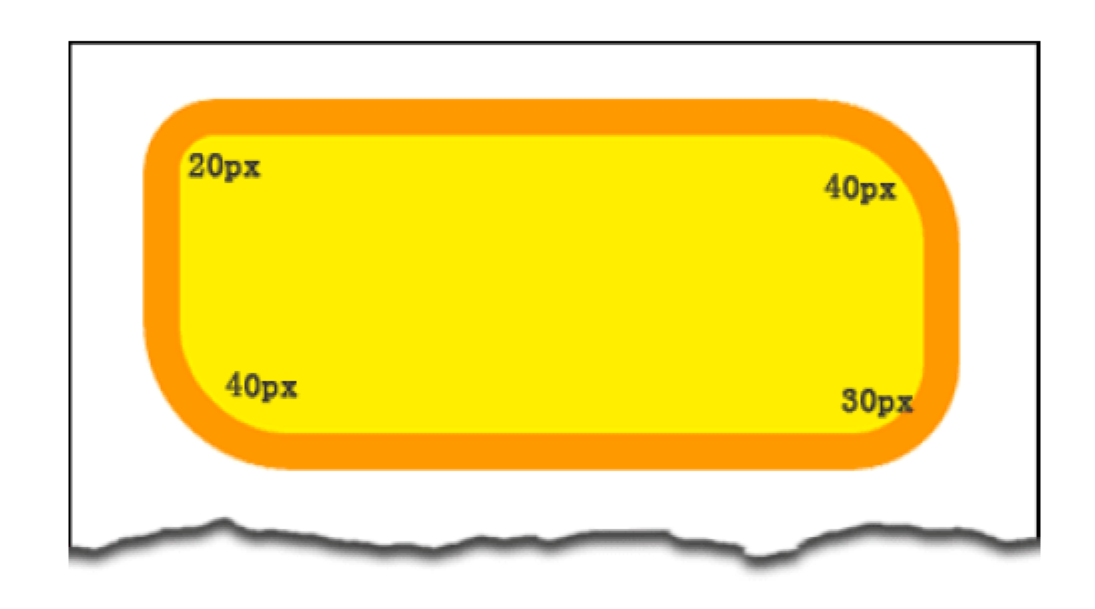


图 3-41　3 个参数的赋值

同样，把 border-radius 属性赋值为 4 个值的集合参数，则第一个值表示左上角的圆角半径，第二个值表示右上角的圆角半径，第三个值表示右下角的圆角半径，第四个值表示左下角的圆角半径。例如，调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
```

```
    background-color:# fe0;
    border: 10px solid #f90;                /* 宽度为 10px 的边框 */
    border-radius:20px 10px 30px 40px;      /* 边框的圆角半径为 20px */
}
</style>
```

运行结果如图 3-42 所示。

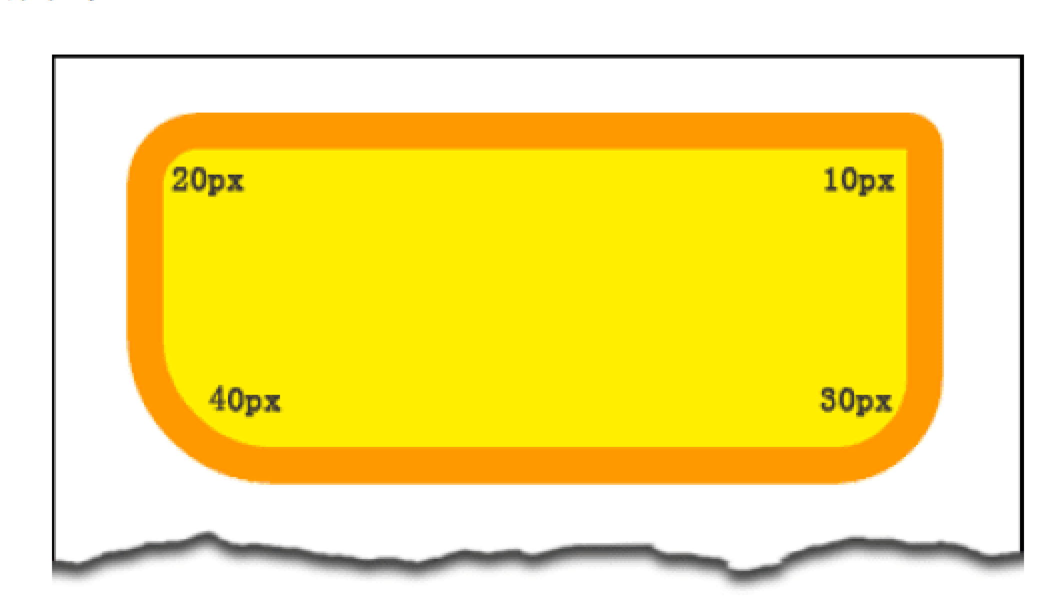


图 3-42　4 个参数的赋值

当然也可以使用 border-radius 属性的 4 个子属性来设计圆角边框。例如，我们把 border-radius 属性赋值为 4 个值的集合参数，改成子属性的实现方法，调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:# fe0;
    border: 10px solid #f90;                /* 宽度为 10px 的边框 */
    border-top-left-radius:20px;            /* 左上角的圆角半径为 20px */
    border-top-right-radius:10px;           /* 右上角的圆角半径为 10px */
    border-bottom-right-radius:30px;        /* 右下角的圆角半径为 30px */
    border-bottom-left-radius:40px;         /* 左下角的圆角半径为 40px */
}
</style>
```

运行结果同样如图 3-42 所示。

4．两组半径的圆角

border-radius 属性还可以为边框的圆角同时指定两组半径值：第一组的值表示圆角的水平半径，第二组的值表示圆角的垂直半径，如图 3-43 所示。第一组和第二组的半径值用“/”隔开，如 border-radius 属性的语法表示。

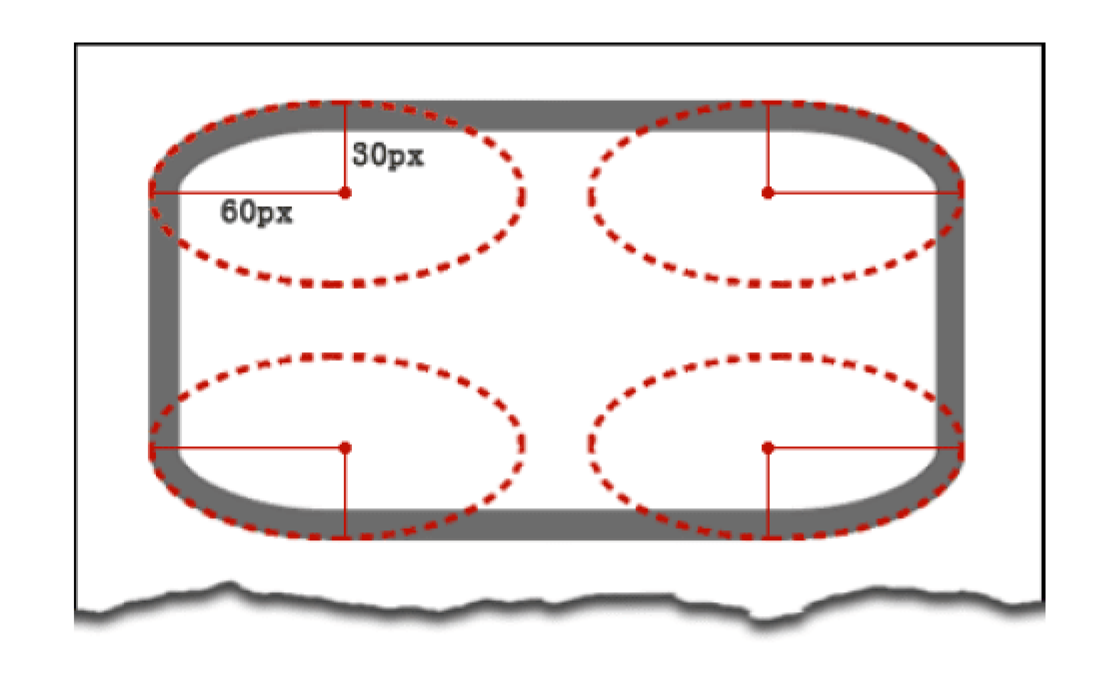


图 3-43　水平半径和垂直半径

如果半径只有一组，则默认为垂直半径等于水平半径，表示这个圆角是一个 1/4 圆；如果圆角的水平半径和垂直半径中任一个为 0，则这个角就变成直角了。下面看一个示例。

【示例 3-22】 两组半径的圆角。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>两组半径的圆角</title>
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:#fe0;
    border: 10px solid #f90;        /* 宽度为 10px 的边框 */
    border-radius:20px/40px;        /* 斜线间隔的两组半径 */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-44 所示。

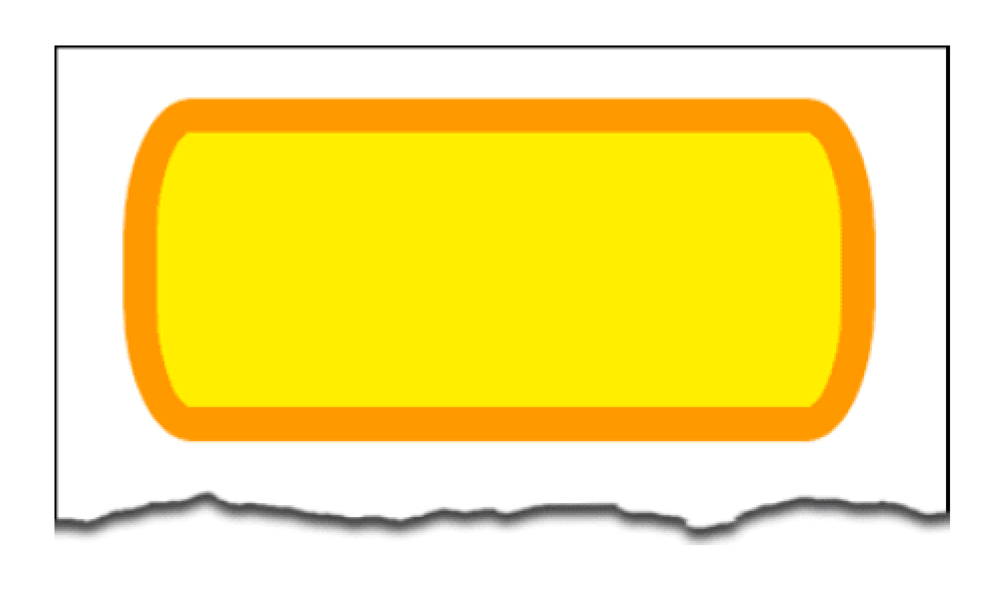


图 3-44　两组半径的圆角

代码分析：在示例 3-22 中，添加 border-radius 属性，并赋值为“20px/40px”，该值由斜线间隔，代表圆角的两组半径（即水平半径和垂直半径）。此时边框的每个圆角是一个 1/4 的椭圆。

当然，也可以赋 border-radius 属性值为两组 4 个值。即第一组的 4 个值，分别表示左上角、右上角、右下角、左下角圆角的水平半径，第二组的 4 个值分别表示左上角、右上角、右下角、左下角圆角的垂直半径。调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    background-color:#fe0;
    border: 10px solid #f90;                        /* 宽度为 10px 的边框 */
    border-radius: 20px 30px 40px 50px/30px 40px 10px 0;
                                                    /* 斜线间隔的两组半径 */
}
</style>
```

运行结果如图 3-45 所示。

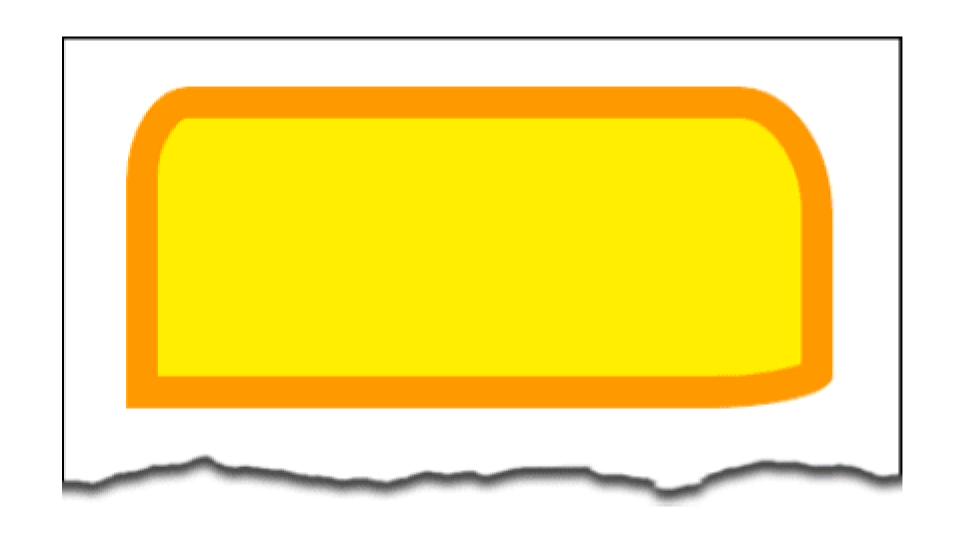


图 3-45　两组半径的圆角

代码分析：在设置第二组半径时，将边框左下角圆角的垂直半径设置为 0，则边框的左下角为直角。根据 CSS 的赋值规则，也可以给 border-radius 属性赋值为：两组 2 个值或两组 3 个值。

提示：值得一提的是，border-radius 属性所派生的 4 个子属性，不能使用两组半径的方法赋值。期待着未来能够获得支持。

3.4.2　设计图像边框——border-image 属性

在 CSS 3 之前，图像不能直接应用于边框，设计者们通常把边框的每个角或每条边单独做一张图，并转而使用背景图像的方式，模拟实现边框，这与边框本身的属性没有一点关系。针对这种曲折繁琐的实现方法，CSS3 中新增了 border-image 属性，专门用于图像边框的处理。它的强大之处更在于它能灵活地分割图像，并应用于边框。

1. 参数说明

border-image 属性的语法如下：

```
border-image : none | <image> [ <number> | <percentage>]{1,4} [ / <border-width>{1,4} ]? [ stretch | repeat | round ]{0,2}
```

取值说明如下。

- ❑ none：默认值。表示边框没有背景图。
- ❑ <image>：使用绝对或相对的 url 地址指定图像源。
- ❑ <number>：裁切边框图像大小；属性值没有单位，默认单位为像素。
- ❑ <percentage>：裁切边框图像大小，使用百分比表示。
- ❑ <border-width>：由浮点数字和单位标识符组成的长度值，不可为负值，用于设置边框宽度。
- ❑ stretch、repeat、round：分别表示拉伸、重复、平铺，其中默认值为 stretch。

2. 派生子属性

border-image 属性是一个复合属性。根据边框方位的不同，可派生如下 8 个特定方位的子属性。

- ❑ border-top-image：定义上边框的图像。
- ❑ border-right-image：定义右边框的图像。
- ❑ border-bottom-image：定义下边框的图像。

- ❑ border-left-image：定义左边框的图像。
- ❑ border-top-left-image：定义左上角边框的图像。
- ❑ border-top-right-image：定义右上角边框的图像。
- ❑ border-bottom-left-image：定义左下角边框的图像。
- ❑ border-bottom-right-image：定义右下角边框的图像。

根据边框图像的处理功能，border-image 属性又派生类以下几个子属性。

- ❑ border-image-source：定义边框的图像源，使用绝对或相对的 url 地址。
- ❑ border-image-slice：定义边框图像的切片，设置图像的边界向内的偏移长度。
- ❑ border-image-repeat：定义边框图像的展现方式：拉伸、重复、平铺。
- ❑ border-image-width：定义图像边框的宽度，也可使用 border-width 属性实现相同的功能。
- ❑ border-image-outset：定义边框图像的偏移位置。

3. 图像的切片

border-image 属性的语法中的<number>或<percentage>都可用于定义边框图像的切片，也可以使用子属性 border-image-slice 来定义边框图像的切片，但子属性 border-image-slice 没有获得任何主流浏览器的支持。

关于 border-image 属性及其子属性 border-image-slice 定义的切片，我们基于如图 3-46 所示的边框素材，说明如下。

- ❑ 切片可使用<number>或<percentage>值定义实现，定义出 9 个切片进行边框图像渲染。
- ❑ <number>为数值，没有单位，默认单位为像素。
- ❑ <percentage>为百分比值，该百分比是相对于图像而言的。
- ❑ 只要定义 4 个数值或百分比值，就会从图像的边界分别在上、右、下、左 4 个方向内偏移相应的长度（如图 3-47 所示的箭头），即可形成 4 条直线（如图 3-47 所示的虚线），通过这 4 条直线，可以确定 9 个切片。
- ❑ 属性会在 4 个方向上设置向内偏移的长度，它遵循 CSS 方位规则，按照上、右、下、左的顺时针方向逐个赋值。当然，两个值和三个值的，都会按照统一的方位规则去解释。
- ❑ 在 9 个切片中，4 个角上的切片会直接显示；4 个边上的切片，则可以设置拉伸、平铺等显示方式；中间的切片，会以元素背景形式显示，其显示方式会与 4 个边上的切片保持一致；所有的切片，都会根据边框的宽度与切片宽度的比例进行缩放。

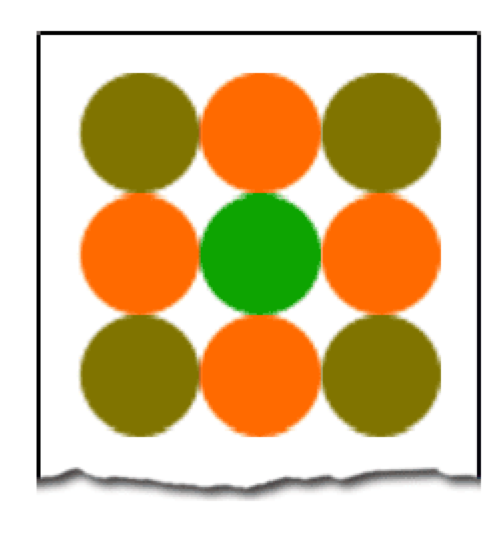


图 3-46　边框图像素材

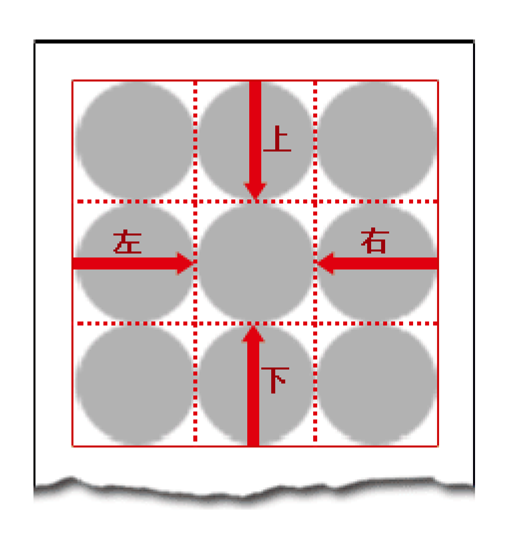


图 3-47　边框图像切片

4. 图像边框示例

下面通过示例来了解图像边框的设计方法，并理解边框图像的切片原理。这里使用图 3-46 所示的边框图像素材，图像尺寸为 90px×90px，图像中的每个圆圈的外切矩形尺寸均为 30px×30px。

【示例 3-23】 设计图像边框。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>设计图像边框</title>
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 30/30px;
                                             /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 30/30px;
                                             /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 30/30px;
                                             /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 30/30px;
                                             /* 标准用法 */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-48 所示。

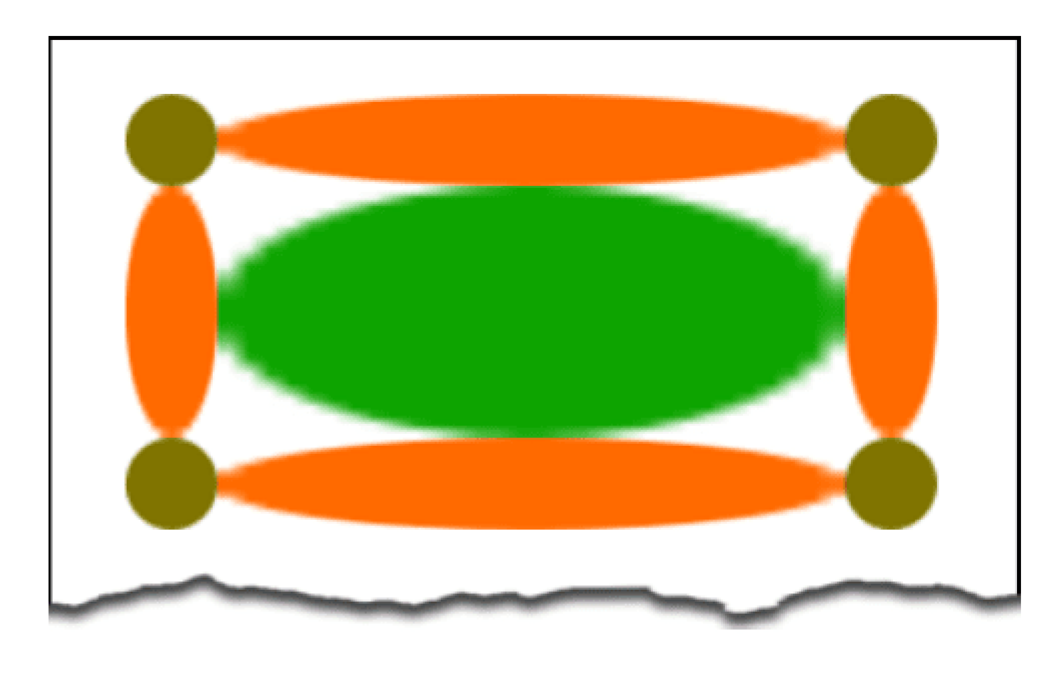


图 3-48　设计的图像边框

代码分析：如示例 3-23 中，仅 border-image 属性即可实现图像边框，并根据不同的浏览器内核给出兼容性的用法。该示例中，设置一个内偏移量值为 30px，图像会在 4 个方向均以 30px 的内偏移量把图像分割成 9 个相等的切片，各位一个圆圈图形。如图 3-48 所示，4 个角直接显示相应的切片，而 4 个边上的切片拉伸显示，中间的切片也以背景的形式拉伸显示。因为在不设置显示方式的时候，默认值为 stretch，即拉伸显示。

遵循 CSS 赋值的方位规则：设置一个偏移值（A）时，表示图像的 4 个方位的向内偏移值均为该值（A）；设置 2 个偏移值（A、B）时，图像的上、右、下、左 4 个方位的内

偏移值分别为 A、B、A、B；设置 3 个偏移值（A、B、C）时，则图像的上、右、下、左 4 个方位的内偏移值分别为 A、B、C、B；设置 4 个偏移值（A、B、C、D）时，则图像的上、右、下、左 4 个方位的内偏移值分别为 A、B、C、D。同样，边框的宽度也遵循这个规则。

所以，示例 3-23 中的样式表也可以是如下写法：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 30 30 30 30/30px;
                                            /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 30 30 30 30/30px;
                                            /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 30 30 30 30/30px;
                                            /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 30 30 30 30/30px;
                                            /* 标准用法 */
}
</style>
```

运行结果不变，如图 3-48 所示。很显然，我们可以把每个方位的内偏移设为不同的值，以调整图像的切片。

例如，我们设置两个不相等的偏移值，产生不同尺寸的切片，调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 40 30/30px;
                                                /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 40 30/30px;
                                                /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 40 30/30px;
                                                /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 40 30/30px;
                                                /* 标准用法 */
}
</style>
```

运行结果如图 3-49 所示，图像的切片如图 3-50 所示。

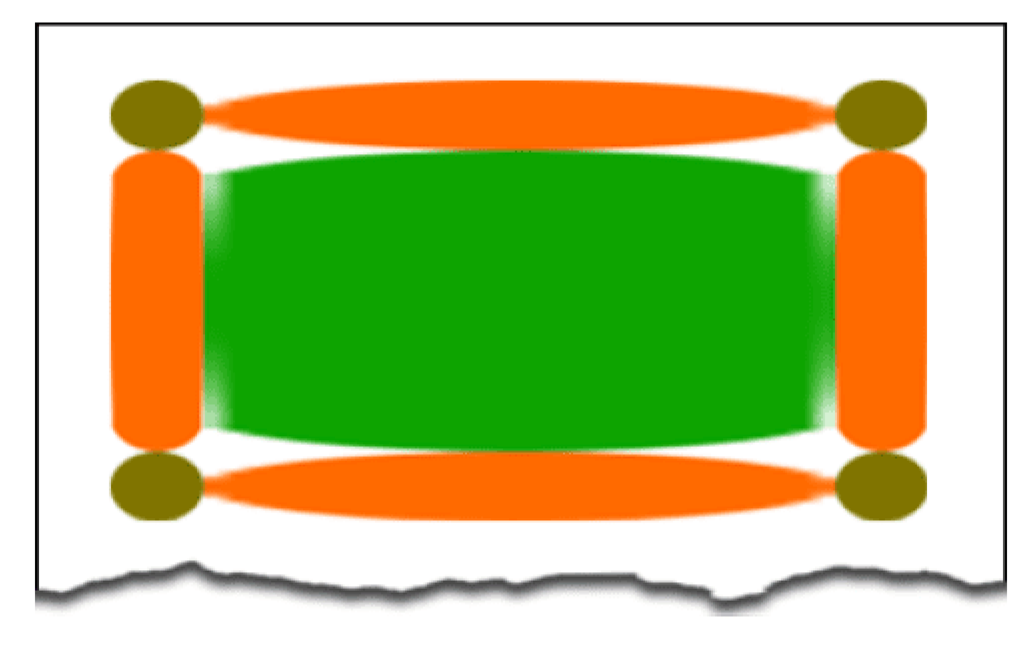


图 3-49　切片不等的图像边框图

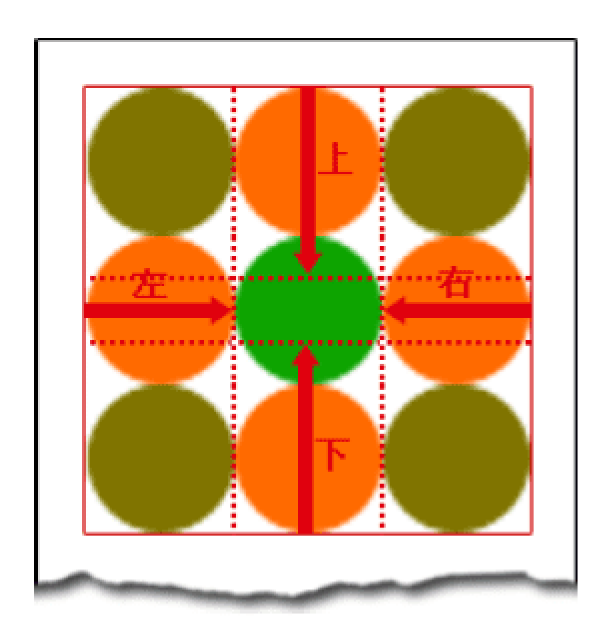


图 3-50　图像的切片

当上下两个方位的内偏移值之和大于或等于图像高度，且左右两个方位的内偏移值之

和也大于或等于图像的宽度时，则只有 4 个角能获得切片，并根据尺寸缩放大小。调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 60 70 50 30/30px;
                                            /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png)  60  70  50  30/30px;
                                            /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png)   60   70   50   30/30px;
                                            /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png)    60    70    50    30/30px;
                                            /* 标准用法 */
}
</style>
```

运行结果如图 3-51 所示。



图 3-51　仅 4 个角有切片图像的边框

如果上下两个方位的内偏移值都大于或等于图像的高度，且左右两个方位的内偏移值也都大于或等于图像的宽度，则 4 个角能获得完整图像的切片。调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 90 100 120 130/
    30px;                                       /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 90 100 120 130/30px;
                                                /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png)  90  100  120  130/30px;
                                                /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png)   90   100   120   130/30px;
                                                /* 标准用法 */
}
</style>
```

则运行结果如图 3-52 所示。

5. 图像切片显示方式

边框图像切片的显示方式对应的子属性为 border-image-repeat，包含三个值：stretch、repeat、round，分别表示拉伸、重复、平铺，其中默认值为 stretch。



图 3-52　整个图像作为切片的边框

【示例 3-24】 边框图像重复显示。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>图像显示方式</title>
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 30/30px repeat;
                                                /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png)  30/30px  repeat;
                                                /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png)   30/30px   repeat;
                                                /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 30/30px repeat;
                                                /* 标准用法 */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-53 所示。

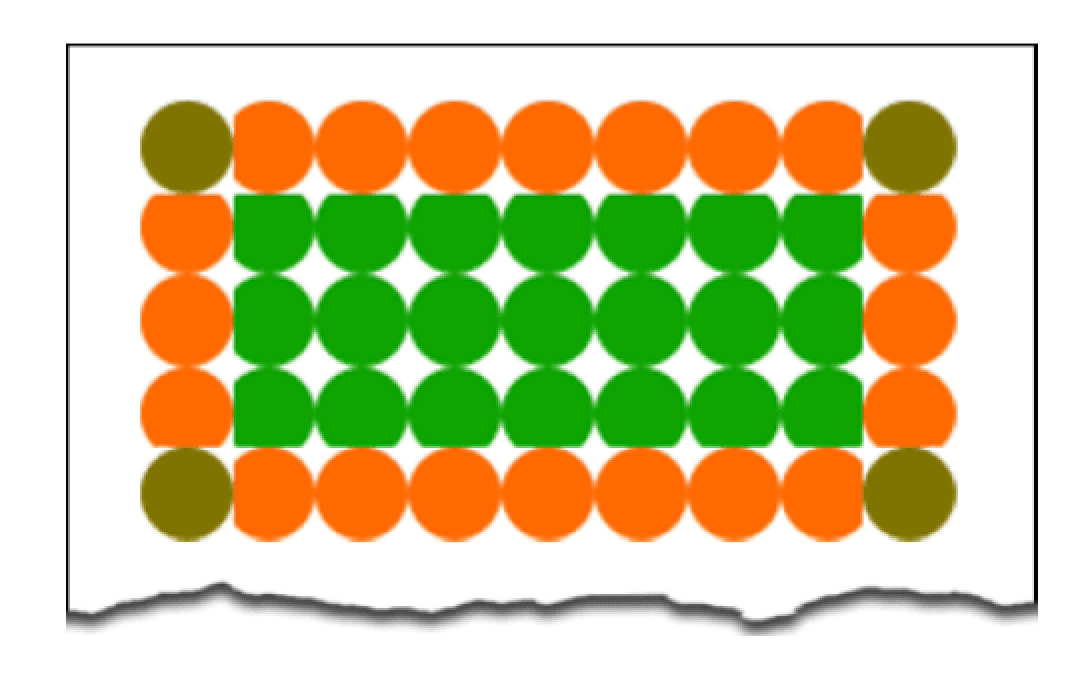


图 3-53　图像显示方式 repeat

代码分析：在示例 3-24 中，指定了边框图像的显示方式为重复（repeat），则 4 个边框的图像重复显示，且重复过程中会保持所属切片的长宽比例不变。如图 3-53 所示，切片

是从边框的中间开始向周围重复平铺的，在边缘区域，可能会被部分隐藏，不能显示完整的切片。

属性值 round 也是把切片重复地平铺，与属性值 repeat 不同的是，在切片平铺过程中，会根据边框的尺寸调整切片的长宽比例，以保证在边缘区域也能显示完整的切片。调整示例 3-24 的样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png)  30/30px  round;
                                            /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 30/30px round;
                                            /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 30/30px round;
                                            /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 30/30px round;
                                            /* 标准用法 */
}
</style>
```

运行结果如图 3-54 所示。

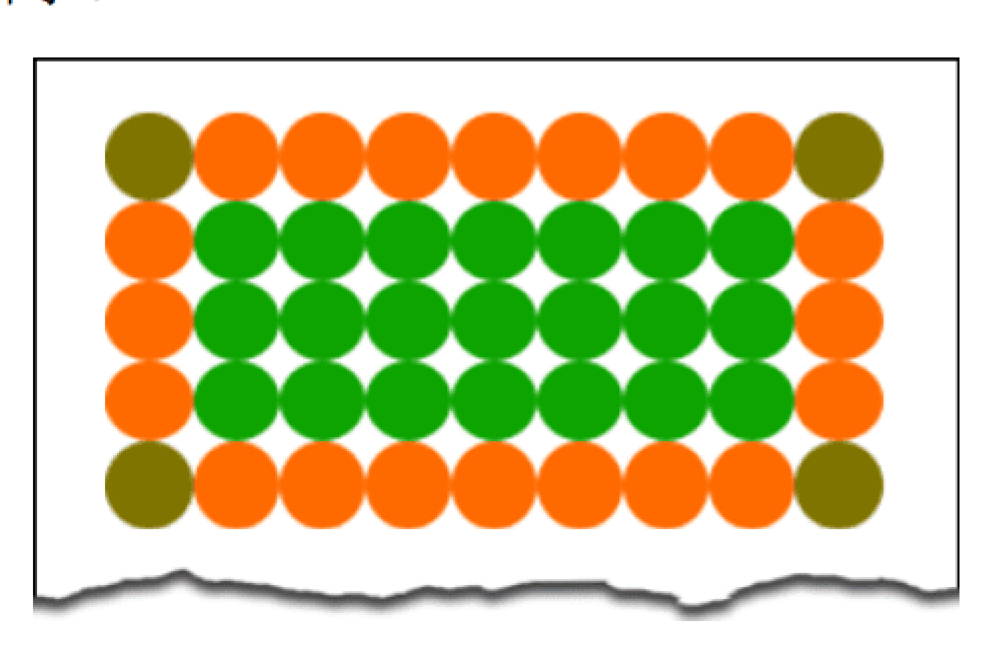


图 3-54　图像显示方式 round

也可以定义显示方式为两个值：第一个值用于指定上、下两个边框的切片显示方式，第二个值用于指定左、右两个边框的切片显示方式。调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png)  30/30px  round
    stretch;                                /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png)  30/30px  round
    stretch;                                /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 30/30px round stretch;
                                            /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png)  30/30px  round  stretch;
                                            /* 标准用法 */
}
</style>
```

运行结果如图 3-55 所示。

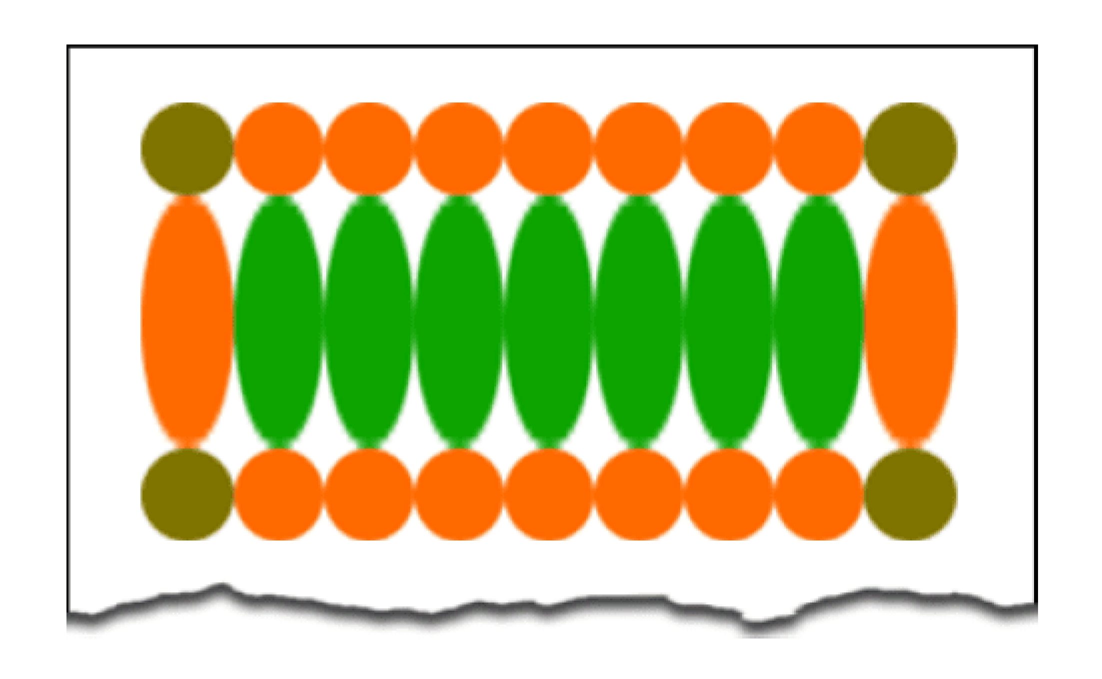


图 3-55　图像显示方式 round+stretch

提示：关于图像切片的显示方式，浏览器允许定义一个值或两个值，并且可以遵循 CSS 赋值的方位规则来解析。但是，不允许同时定义 3 个值或 4 个值。

6. 图像边框的宽度

边框的宽度可以在 border-image 属性中设置，也可以使用 border-width 属性来指定宽度。图像的切片会根据边框的尺寸自动缩放切片。

【示例 3-25】 边框的宽度。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>图像边框的宽度</title>
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png)  30/10px  round;
                                                  /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 30/10px round;
                                                  /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png) 30/10px round;
                                                  /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png) 30/10px round;
                                                  /* 标准用法 */
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-56 所示。

代码分析：在示例 3-25 中，设置了一个值为 10px 的边框宽度，即同时定义了 4 个边框的宽度。如图 3-56 所示，图像的切片缩小了很多。图像边框的宽度设置，也遵循 CSS 赋值的方位规则，可以为不同的边定义不同的宽度。调整样式表如下：

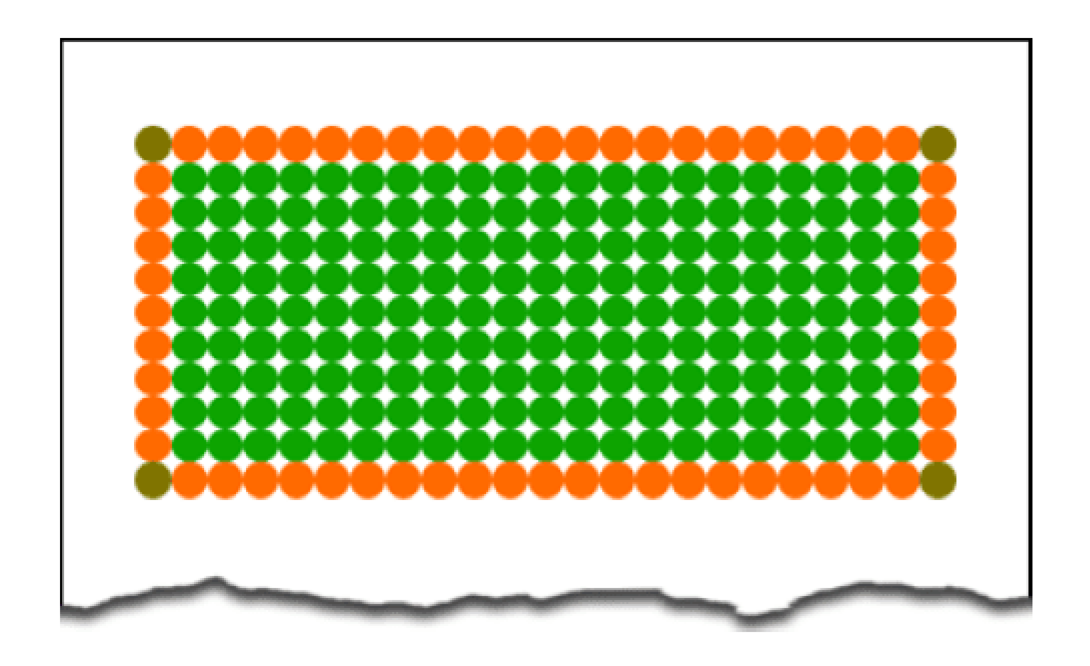


图 3-56　图像边框宽度为 10px

```
<style type="text/css">
div {
    width:200px;
    height:80px;
    -webkit-border-image:url(../images/borderimage.png) 30/30px 20px
    round;                                          /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage.png) 30/30px 20px round;
                                                    /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage.png)  30/30px  20px  round;
                                                    /* 兼容 presto 内核 */
    border-image:url(../images/borderimage.png)   30/30px   20px   round;
                                                    /* 标准用法 */
}
</style>
```

运行结果如图 3-57 所示。

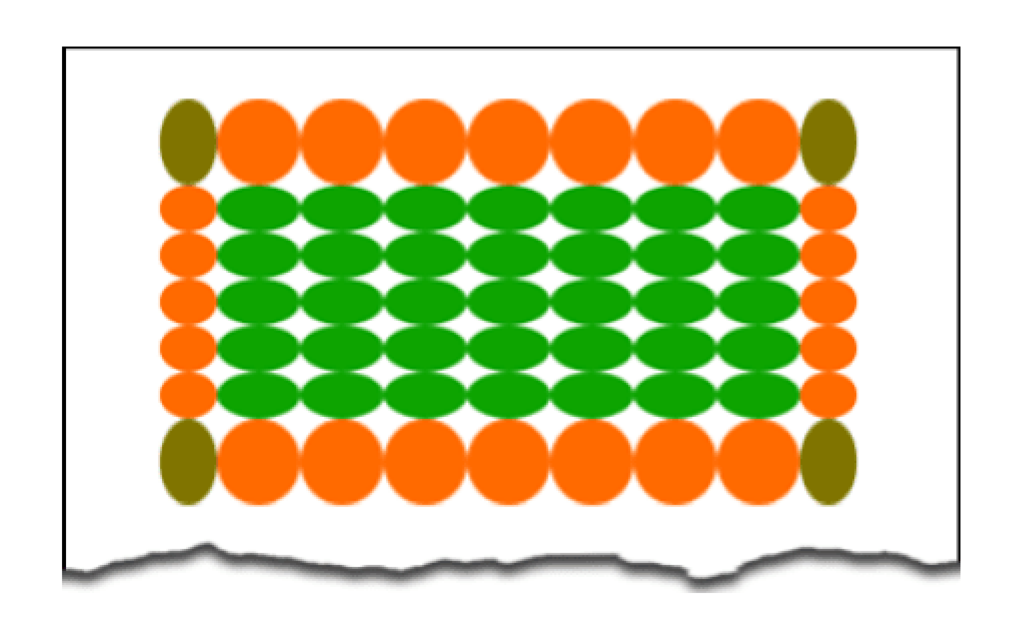


图 3-57　图像显示方式 round+stretch

提示：本节的示例图全部是火狐浏览器中的显示效果。不同内核的浏览器，解析 border-image 属性有所区别，但是火狐浏览器的支持效果更贴近 CSS 3 规范。

3.4.3　设计多色边框——border-color 属性

border-color 属性用于设置边框的颜色，在之前的 CSS 规范中已经定义了。不过，CSS 3 增强了该属性的功能，可以用它为边框设置更多的颜色，可以直接设计出丰富的边框效果。

1. 参数说明

border-color 属性的语法如下：

```
border-color : [ <color> | transparent ] {1,4}
```

取值说明如下。

- <color>：是一个颜色值，可以是半透明颜色。
- transparent：透明值。不设置边框颜色时，默认为该值。
- border-color 属性，遵循 CSS 赋值的方位规则，可以分别为元素的 4 个边框设置颜色。

2. 派生子属性

border-color 属性本身可定义 1～4 种颜色，用于设置各个边框的颜色，但无法同时为边框指定多种颜色，因为这会导致歧义。border-color 属性派生 4 个子属性，分别用于 4 个边框颜色的设置，如下。

- border-top-color：定义元素顶部边框的颜色。
- border-right-color：定义元素右侧边框的颜色。
- border-bottom-color：定义元素底部边框的颜色。
- border-left-color：定义元素左侧边框的颜色。

这 4 个子属性分别为各个边框指定颜色，可以指定多种颜色。只是指定多种颜色的功能，仅火狐浏览器有私有属性支持。

3. 多色边框示例

为边框指定多种颜色，需要使用 border-color 属性的子属性来定义。border-color 属性本身不能为一个边框定义多种颜色，因为会带来歧义。

【示例 3-26】 为边框定义多种颜色。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>为边框定义多种颜色</title>
<style type="text/css">
div {
    height:160px;
    border:20px solid #999;
    /* 兼容 gecko 内核的浏览器如火狐 */
    -moz-border-top-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90 #fa0;
    -moz-border-right-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90
    #fa0;
    -moz-border-bottom-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90
    #fa0;
    -moz-border-left-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90
    #fa0;
    /* 标准用法 */
    border-top-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90 #fa0;
    border-right-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90 #fa0;
    border-bottom-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90 #fa0;
    border-left-colors:#f10 #f20 #f30 #f40 #f50 #f60 #f70 #f80 #f90 #fa0;
}
</style>
<body>
<div></div>
</body>
</html>
```

运行结果如图 3-58 所示。

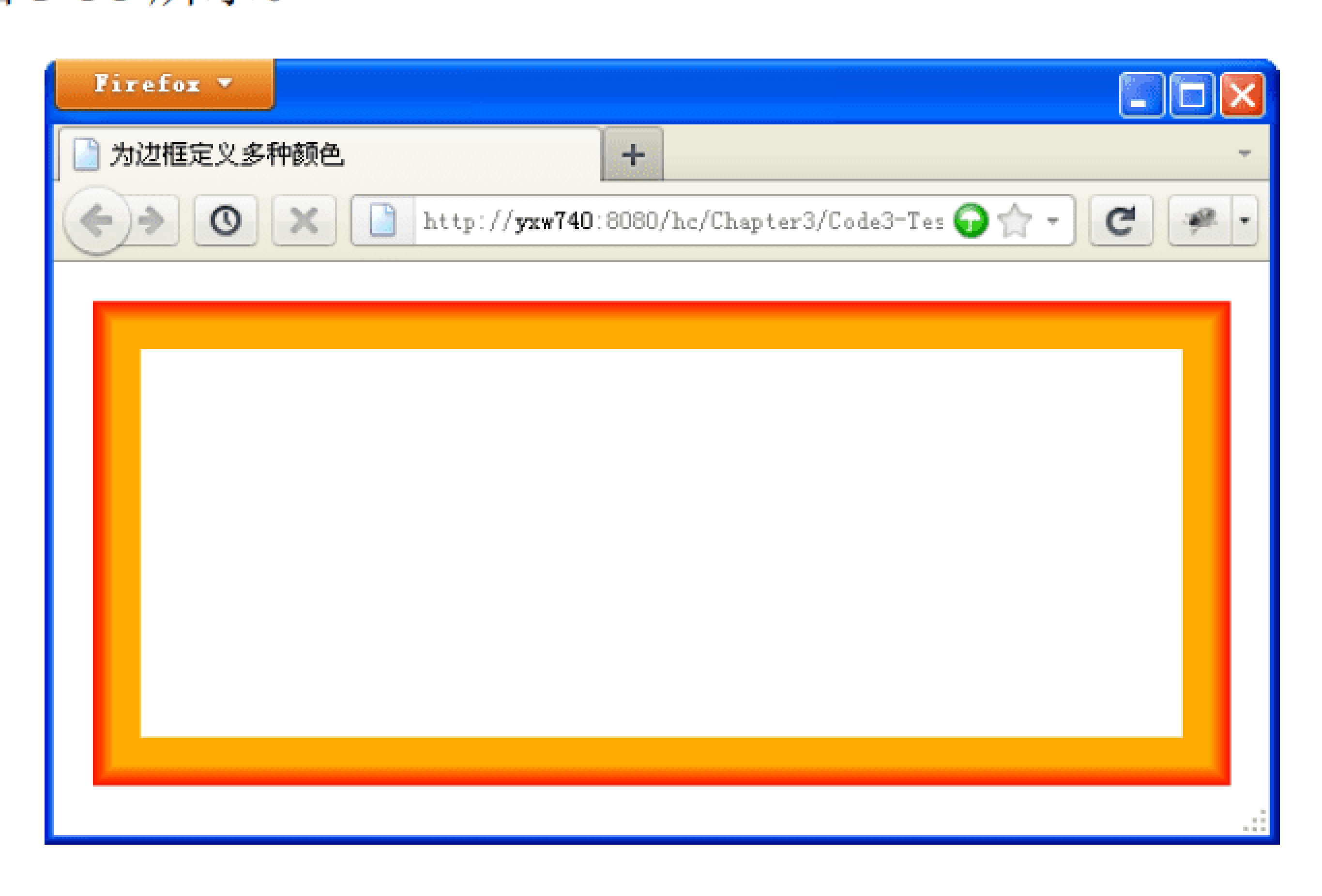


图 3-58　多色边框在 Firefox 中的显示效果

代码分析：在示例 3-26 中，为每个边框分别定义了 10 种颜色。如图 3-58 所示，颜色从外到内显示，每种颜色只占有 1 像素的宽度。本示例中，边框宽度为 20px，10 种颜色中的最后一种颜色，将被用于剩下的宽度。

3.4.4　实验室：使用新技术设计网页

CSS 3 的圆角与图像边框有着强大的功能，灵活使用圆角和边框属性，可以设计出各种各样的页面效果。本节介绍一个使用新技术设计的网页案例，该案例综合运用了 border-radius 属性、border-image 属性和 border-color 属性。

1. 案例简介

本节所介绍的案例，是一个简易网页，包括导航部分、搜索部分、图片切换预览部分和功能按钮部分。其中会涉及大量的圆角和图像修饰的边框。整个案例，将会使用两个图片素材，用于设计图像边框，如图 3-59 所示。其中也用到了渐变属性，省了不少背景图片，渐变属性将在后面章节里讲解。

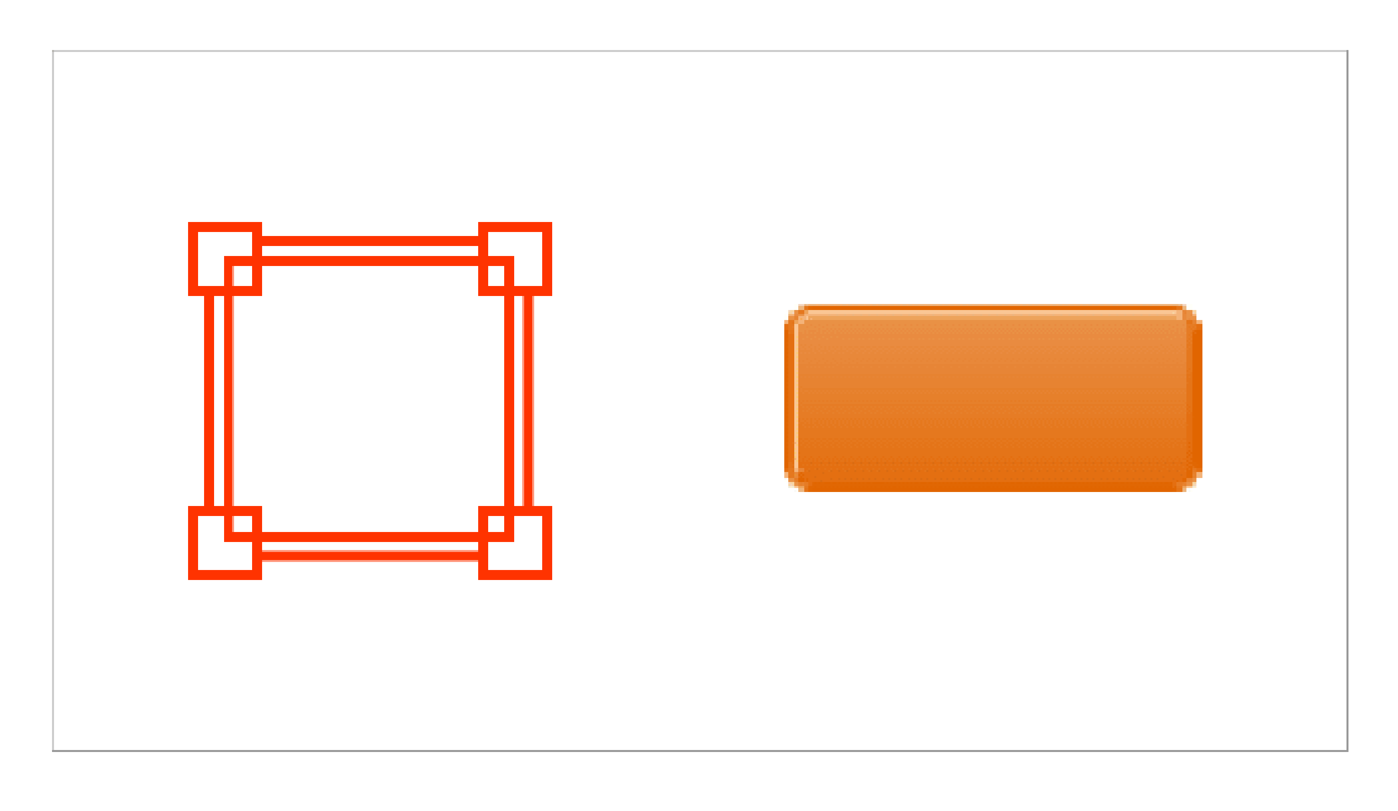


图 3-59　图像边框素材

整个案例演示效果如图 3-60 所示。



图 3-60　使用新技术设计的网页

2. 设计网页元素

设计网页元素结构，包含导航、搜索、图片预览、功能按钮等部分。

【示例 3-27】 使用新技术设计的网页。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>使用新技术设计的网页</title>
</head>
<body>
<!--显示区域-->
<div id="area">
  <!--导航部分-->
  <ul id="nav">
    <li class="cur">风景</li>
    <li>人物</li>
    <li>素材</li>
  </ul>
  <!--搜索部分-->
  <div id="search">
    <input type="text" name="key" value="" />
    <input type="submit" value="GO" />
  </div>
  <!--图片切换预览部分-->
  <div id="box">
    <table width="100%" border="0" align="center" cellpadding="0"
    cellspacing="0">
      <tr>
        <td><div class="prev"><a href="#">3</a></div></td>
        <td class="pics"><img src="../images/Iceland.jpg" /></td>
        <td><div class="next"><a href="#">4</a></div></td>
      </tr>
    </table>
```

```
  </div>
  <!--功能按钮部分-->
  <div id="but"><a class="first" href="#">分享</a><a href="#">转发</a><a
class="last" href="#">收藏</a>
  <span class="more">列表模式</span></div>
</div>
</body>
</html>
```

下面，我们一步一步地给各个功能块添加样式表，并添加到示例 3-27 中。

3. 设计显示区域的边框

使用预先准备的图片素材，设计一个图像边框的样式表。

```
<style type="text/css">
#area {
    width:480px;     /* 设置宽度 */
    padding:10px;
    margin:0 auto;    /* 元素本身居中 */
    -webkit-border-image:url(../images/borderimage2.png) 15/15px;
                                                /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/borderimage2.png) 15/15px;
                                                /* 兼容 gecko 内核 */
    -o-border-image:url(../images/borderimage2.png) 15/15px;
                                                /* 兼容 presto 内核 */
    border-image:url(../images/borderimage2.png) 15/15px;
                                                /* 标准用法 */
}
</style>
```

4. 设计导航部分

导航部分使用的是列表元素，设计成横向排列显示，鼠标经过时，背景变成红色，内容文字变为白色。

```
<style type="text/css">
#nav {
    list-style:none;            /* 不使用列表样式 */
    margin:0;                   /* 设置外边距 */
    padding:0 60px;             /* 设置内边距 */
    font-size:12px;
    height:20px;                /* 高度控制 */
    margin-bottom:-2px;         /* 实际内容下沉 2 像素，紧贴下面的搜索框 */
}
#nav li {
    height:20px;                /* 高度控制 */
    line-height:20px;           /* 行高控制 */
    padding:0 10px;
    margin-left:2px;
    float:left;                 /* 元素向左浮动，列表变成横向排列显示 */
    cursor:pointer;             /* 手型指针 */
}
#nav .cur, #nav li:hover {
    background-color:#ff3333;   /* 鼠标经过，背景变为红色 */
    border-radius:3px 3px 0 0;  /* 左上角和右上角设计为小圆角 */
```

```
    color:#FFF;                /* 鼠标经过，文字变为白色 */
}
</style>
```

5. 设计搜索部分

搜索部分使用的是文本框和提交按钮，把它们都设计成部分角是圆角，并紧贴在一起。

```
<style type="text/css">
#search {
    text-align:center;
}
#search input {
    border:1px solid #ff3333;       /* 均使用红色边框 */
}
#search input[type=text] {          /* 设置文本框样式 */
    border-radius:10px 0 0 10px;    /* 左上角和左下角使用圆角，圆角半径为 10px */
    border-right-width:0;           /* 右边框宽度重新设置为 0 */
    padding-left:10px;              /* 左内边距 10px */
    width:380px;                    /* 超长宽度 */
}
#search input[type=submit] {        /* 设置提交按钮样式 */
    border-radius:0 10px 10px 0;    /* 右上角和右下角使用圆角，圆角半径为 10px */
    margin-left:-10px;
                            /* 消除与文本框的距离，视觉上与文本框衔接在一起 */
    background-image: -moz-linear-gradient(top, #ffffcc, #ffcc99);
                                    /*渐变*/
    background-image: -webkit-gradient(linear, left top, left bottom,
    from(#ffffcc), to(#ffcc99));    /* 渐变 */
}
</style>
```

6. 设计图片预览部分

图片预览部分包括图片预览和切换图片按钮：上一张图和下一张图。把图片设计成四个角是圆角，其后按钮设计成部分圆角。

```
<style type="text/css">
#box {
    padding-top:10px;
}
#box .pics {
    text-align:center;
}
#box .pics img {
    width:400px;                        /* 固定图像宽度 */
    -webkit-border-radius:0 20px;       /* 设计圆角图片 */
    -moz-border-radius:0 20px;          /* 设计圆角图片*/
    -o-border-radius:0 20px;            /* 设计圆角图片*/
    border-radius:10px;                 /* 设计圆角图片*/
}
#box .prev, #box .next {
    width:25px;
    height:25px;
    float:left;
    line-height:25px;
```

```
    text-align:center;
    vertical-align:middle;                /* 垂直方向居中 */
    background-color:#f00;                /* 背景颜色为红色 */
    font-family:webdings;                 /* 图形字体 */
}
#box .prev:hover, #box .next:hover {
    /* 鼠标经过，切换按钮的背景变成渐变背景 */
    background-image: -moz-linear-gradient(top, #ff3300, #663333);
    background-image: -webkit-gradient(linear, left top, left bottom,
    from(#ff3300), to(#663333));
}
#box .prev {
    border-radius:45px 0 0 45px;          /* 左边切换按钮的圆角 */
}
#box .next {
    border-radius:0 45px 45px 0;          /* 右边切换按钮的圆角*/
}
#box .prev a, #box .next a {
    display:block;
    color:#FFF;
    text-decoration:none;
}
</style>
```

7. 设计功能按钮部分

功能按钮部分包括三个链接按钮和一个模式切换按钮。把三个链接按钮连在一起的形状设计成圆角；模式切换按钮，用预先准备好的图片，设计成图片边框。

```
<style type="text/css">
#but {
    height:20px;
    font-size:12px;
    padding:0 40px;                   /* 调整内边距*/
}
#but a {
    display:block;                    /* 盒模型结构显示 */
    float:left;                       /* 向左浮动 */
    background-image: -moz-linear-gradient(top, #ffffcc, #ffcc99);
                                      /* 渐变 */
    background-image: -webkit-gradient(linear, left top, left bottom,
    from(#ffffcc), to(#ffcc99));      /* 渐变 */
    padding:0 10px;
    height:20px;
    line-height:20px;
    text-align:center;
    color:#333;
    text-decoration:none;
    border-style:solid;               /* 边框样式 */
    border-color:#f90;                /* 边框颜色 */
    border-width:1px 0;               /* 只有上下两个边有边框 */
}
#but .first {
    border-radius:10px 0 0 10px;      /* 第一个链接按钮，左上角和左下角设置为圆角 */
    border-width:1px;                 /* 4 个边有边框 */
}
#but .last {
```

```
    border-radius:0 10px 10px 0;    /* 第三个链接按钮，右上角和右下角设置为圆角 */
    border-width:1px;               /* 4 个边有边框 */
}
#but a:hover {
/* 鼠标经过，切换链接按钮的背景变成渐变 */
    background-image: -moz-linear-gradient(top, #ff3300, #663333);
    background-image:  -webkit-gradient(linear, left top, left bottom,
    from(#ff3300), to(#663333));
    color:#FFF;
}
#but .more {
    float:right;                    /* 向右浮动 */
                                    /* 设置图像边框 */
    -webkit-border-image:url(../images/button.png) 10/5px;
                                    /* 兼容 webkit 内核 */
    -moz-border-image:url(../images/button.png) 10/5px;
                                    /* 兼容 gecko 内核 */
    -o-border-image:url(../images/button.png) 10/5px;
                                    /* 兼容 presto 内核 */
    border-image:url(../images/button.png) 10/5px; /* 标准用法 */
    padding:0 10px;
    color:#FFF;
    cursor:pointer;
}
</style>
```

3.5 小　结

本章集中讲解了最常用、最基础的新增样式表属性。重点讲解了文本阴影、色彩模式、背景和边框 4 个方面的内容。其中新增的 3 个背景属性不容易理解，需要多练习、多琢磨；为圆角设置两组半径，可能不易理解；图像边框所包括的众多子属性，也比较复杂，是个难点。

本节所涉及的内容虽然是基础，但是真正做到灵活运用还是需要下一番功夫的。下一章将介绍 CSS 3 特有的新型弹性盒模型。

3.6 习　题

【习题 1】举例说明一下文本的描边效果和发光效果是如何实现的。

【习题 2】请选择可用于页面设计的合法的颜色值（多选）：

A．#F00　　B．hsl(0, 0%, 90%)　　C．hsla(40, 50%, 50%, 0.8)

D．rgba(255, 153, 0, 0.1)　　E．rgb(255, 153, 0)　　F．hsb(0, 80%, 90%)

【习题 3】请选择 CSS 3 中新增的与背景有关的属性（多选）：

A．background-image　　B．background-origin　　C．background-clip

D．background-repeat　　E．background-size　　F．background-position

【习题 4】请选择用于设置圆角的属性（单选）：

A．border-collapse　　B．border-spacing　　C．border-style

D．border-radius　　E．border-image

第 4 章　灵活的盒布局和界面设计

在 CSS 2.0 的时代，页面布局比较流行 DIV+CSS，但其中的浮动布局有很多不便和缺陷。盒子元素的阴影效果还要借助图片来实现；界面美化方面也略显复杂。对于这些问题，在传统的设计中，没有那么直接的实现，要绕一些弯子。而在 CSS 3 中，这些问题将变得简单而直接。弹性布局直接解决页面布局的问题；盒子阴影有专门的属性处理；友好的界面设计也少走了很多弯路。本章将针对 CSS 3 新增的属性内容进行详细讲解。

4.1　灵活的盒布局

为了解决传统布局中的不足，CSS 3 新增了新型的盒布局。使用盒布局，可以实现盒元素内部的多种布局，包括排列方向、排列顺序、空间分配和对齐方式等，大大增强了布局的灵活性。这一革命性的改进，大大提高了前端设计师的工作效率和工作水平，下面逐步学习盒布局。

4.1.1　开启盒布局

盒布局是 CSS 3 发展的新型布局方式，它比传统的浮动布局方式更加完善、更加灵活，而使用方法却极为简单。

开启盒布局的方法，就是把 display 属性值设置为 box 或 inline-box。目前没有浏览器支持 box 属性值，为了能兼容 webkit 内核和 gecko 内核的浏览器，可分别使用-webkit-box 和-moz-box 属性。开启盒布局后，文档就会按照盒布局默认的方式来布局子元素。

接下来看一个简单的网页，从左至右排列三个栏目：菜单栏、内容栏和工具栏。传统的实现方式会使用浮动布局方式，现在我们用盒布局的方式来实现三个栏目的横向排列。

【示例 4-1】 一个简单的三栏网页。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>一个简单的三栏网页</title>
<style type="text/css">
.container {
    /* 开启盒布局 */
    display:-webkit-box;                    /* 兼容 webkit 内核 */
    display:-moz-box;                       /* 兼容 gecko 内核 */
    display:box;                            /* 定义为盒子显示 */
```

```
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;              /* 左侧菜单栏背景颜色 */
}
.container .center-content {
    background-color:#390;              /* 中间内容栏背景颜色 */
    width:200px;
}
.container .right-aside {
    background-color:#039;              /* 右侧工具栏背景颜色 */
}
</style>
<body>
<div class="container">
  <div class="left-aside">
    <h2>菜单</h2>
    <ul>
      <li>HTML5</li>
      <li>CSS 3</li>
      <li>活动沙龙</li>
      <li>研发小组</li>
    </ul>
  </div>
  <div class="center-content">
    <h2>内容</h2>
    <p>盒布局是 CSS 3 发展的新型布局方式，它比传统的浮动布局方式更加完善、更加灵活，而
    使用方法却极为简单。</p>
    <p>开启盒布局方法，就是设置 display 属性值为 box（或 inline-box）。</p>
  </div>
  <div class="right-aside">
    <h2>工具</h2>
    <ul>
      <li>天气预报</li>
      <li>货币汇率</li>
    </ul>
  </div>
</div>
</body>
</html>
```

运行结果如图 4-1 所示。

代码分析：在示例 4-1 中，设置了 container 类的属性“display:box;”，并针对 webkit 内核和 gecko 内核设置了各自的私有属性值。接下来，被赋予 container 类的元素的内部元素，将改变原有的文档流动方式，使用盒布局默认的文档流动方式，如图 4-1 所示。

盒布局包含多方面的内容，开启盒布局只是盒布局的第一步。



图 4-1　一个简单的三栏网页

4.1.2　元素的布局方向——box-orient 属性

CSS 3 新增的 box-orient 属性，可用于定义盒元素的内部布局方向。基于 webkit 内核的替代私有属性是-webkit-box-orient，基于 gecko 内核的替代私有属性是-moz-box-orient。

1. 参数说明

box-orient 属性的语法如下：

```
box-orient : horizontal | vertical | inline-axis | block-axis | inherit
```

取值说明如下。

- ❑ horizontal：表示盒子元素在一条水平线上从左到右编排它的子元素。
- ❑ vertical：表示盒子元素在一条垂直线上从上到下编排它的子元素。
- ❑ inline-axis：默认值，表示盒子元素沿着内联轴编排它的子元素，表现为横向编排。
- ❑ block-axis：表示元素沿着块轴编排它的子元素，表现为垂直编排。
- ❑ inherit：表示继承父元素中 box-orient 属性的值。

2. 示例介绍

基于示例 4-1，在盒布局的方式下，改变三个栏目的布局方向为竖向显示。改变样式表如下。

【示例 4-2】 竖向显示三个栏目。

```
<style type="text/css">
.container {
    display:-webkit-box;                    /* 兼容 webkit 内核 */
    display:-moz-box;                       /* 兼容 gecko 内核 */
    display:box;                            /* 定义为盒子显示 */
    /* 布局方向设置为竖直方向 */
    -webkit-box-orient:vertical;            /* 兼容 webkit 内核 */
    -moz-box-orient:vertical;               /* 兼容 gecko 内核 */
    box-orient:vertical;                    /* 定义为竖向编排显示 */
```

```
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;
}
.container .center-content {                /* 去除了宽度设置 */
    background-color:#390;
}
.container .right-aside {
    background-color:#039;
}
</style>
```

运行结果如图 4-2 所示。



图 4-2　竖向显示三个栏目

代码分析：示例 4-2 中，把 box-orient 属性值设置为 vertical，表示垂直方向布局，并设置了兼容样式。为了显示整齐，同时也取消了内容栏目的宽度设置。由于是盒布局的方式，三个栏目将竖向显示，如图 4-2 所示。

4.1.3　元素的布局顺序——box-direction 属性

在盒布局下，可以设置盒元素内部的顺序为正向或者反向。CSS 3 新增的 box-direction

属性，可用于定义盒元素的内部布局顺序。基于 webkit 内核的替代私有属性是 -webkit-box-direction，基于 gecko 内核的替代私有属性是-moz-box-direction。

1. 参数说明

box- direction 属性的语法如下：

```
box-direction : normal | reverse | inherit;
```

取值说明如下。

- normal：默认值，正常顺序。垂直布局的盒元素中，内部子元素从左到右排列显示；水平布局的盒元素中，内部子元素从上到下排列显示。
- reverse：反向。表示盒元素内部的子元素的排列显示顺序与 normal 相反。
- inherit：表示继承父元素中 box-direction 属性的值。

2. 示例介绍

基于示例 4-1，在盒布局的方式下，把三个栏目改为水平方向上的反向布局。调整样式表如下。

【示例 4-3】 反向显示三个栏目。

```
<style type="text/css">
.container {
    display:-webkit-box;
    display:-moz-box;
    display:box;
    -webkit-box-orient:horizontal;
    -moz-box-orient:horizontal;
    box-orient:horizontal;
    /* 布局顺序属性设置为反向 */
    -webkit-box-direction:reverse;          /* 兼容 webkit 内核 */
    -moz-box-direction:reverse;             /* 兼容 gecko 内核 */
    box-direction:reverse;                  /* 定义为反向顺序 */
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;
}
.container .center-content {
    background-color:#390;
    width:200px;
}
.container .right-aside {
    background-color:#039;
}
</style>
```

运行结果如图 4-3 所示。



图 4-3　反向显示三个栏目

代码分析：示例 4-3 中，把 box- direction 属性值设置为 reverse，表示反向布局顺序，并设置了兼容样式。同时也设置了 box-orient 属性值为 horizontal，表示水平方向布局。在盒布局的方式下，三个栏目将在水平方向上反向显示，如图 4-3 所示。

4.1.4　调整元素的位置——box-ordinal-group 属性

CSS 3 新增的 box-ordinal-group 属性，用于定义盒元素内部的子元素的显示位置。基于 webkit 内核的替代私有属性是-webkit-box-ordinal-group，基于 gecko 内核的替代私有属性是-moz-box-ordinal-group。

1. 参数说明

box-ordinal-group 属性的语法如下：

```
box-ordinal-group: <integer>;
```

取值说明如下。

<integer>：一个自然整数，从 1 开始，表示子元素的显示位置。子元素将根据这个值重新排序，值相等的，将取决于源代码的顺序。子元素的默认值均为 1，按照源代码的位置顺序进行排列。

2. 示例介绍

基于示例 4-1，在盒布局的方式下，调整菜单栏和工具栏的显示位置。调整样式表如下。

【示例 4-4】 调整子元素的显示位置。

```
<style type="text/css">
.container {
    display:-webkit-box;
    display:-moz-box;
    display:box;
```

```
    /* 定义为横向编排显示 */
    -webkit-box-orient:horizontal;          /* 兼容 webkit 内核 */
    -moz-box-orient:horizontal;             /* 兼容 gecko 内核 */
    box-orient:horizontal;                  /* 标准用法 */
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;
    /* 设置菜单栏的位置为 2*/
    -webkit-box-ordinal-group:2;            /* 兼容 webkit 内核 */
    -moz-box-ordinal-group:2;               /* 兼容 gecko 内核 */
    box-ordinal-group:2;                    /* 标准用法 */
}
.container .center-content {
    background-color:#390;
    width:200px;
}
.container .right-aside {
    background-color:#039;
    /* 设置工具栏的位置为 3*/
    -webkit-box-ordinal-group:3;            /* 兼容 webkit 内核 */
    -moz-box-ordinal-group:3;               /* 兼容 gecko 内核 */
    box-ordinal-group:3;                    /* 标准用法 */
}
</style>
```

运行结果如图 4-4 所示。



图 4-4　经过调整的显示顺序

代码分析：在示例 4-4 中，把菜单栏的 box-ordinal-group 属性设置为 2，把工具栏的 box-ordinal-group 属性设置为 3，以改变三个栏目的显示顺序。如果 4-4 所示，三个栏目按照预定的顺序显示。

4.1.5　弹性空间分配——box-flex 属性

CSS 3 新增的 box-flex 属性，用于定义盒元素内部的子元素是否具有空间弹性。当盒元素的尺寸缩小（或扩大）时，被定义为有空间弹性的子元素的尺寸也会缩小（或扩大）。每当盒元素有额外的空间时，具有空间弹性的子元素，会扩大自身大小来填补这一空间。基于 webkit 内核的替代私有属性是-webkit-box-flex，基于 gecko 内核的替代私有属性是-moz-box-flex。

1. 参数说明

box-flex 属性的语法如下：

```
box-flex: <value>;
```

取值说明如下。

<value>：该属性值是一个整数或者小数，不可为负值，默认值为 0.0。使用空间弹性属性设置，使得盒元素的内部元素的总宽度和总高度，始终等于盒元素的宽度与高度。不过只有当盒元素具有确定的宽度或高度时，才能表现出子元素的空间弹性。

2. 子元素的弹性空间

下面的示例中，在盒元素的内部设置了宽度相等的三栏：菜单、文章列表和工具，并设置菜单栏的 box-flex 属性，使其具有空间弹性。

【示例 4-5】 具有空间弹性的菜单栏。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>具有空间弹性的菜单栏</title>
<style type="text/css">
.container {
    width:100%;                    /* 设置盒元素的宽度为 100% */
    background-color:#CCC;
    display:-webkit-box;
    display:-moz-box;
    display:box;
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
    width:100px;                   /* 设置三个栏目的固定宽度 100px */
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;
    /* 设置菜单栏具有空间弹性 */
```

```
    -webkit-box-flex:1;            /* 兼容 webkit 内核 */
    -moz-box-flex:1;               /* 兼容 gecko 内核 */
    box-flex:1;                    /* 标准用法 */
}
.container .center-content {
    background-color:#390;
}
.container .right-aside {
    background-color:#039;
}
</style>
<body>
<div class="container">
  <div class="left-aside">
    <h2>菜单</h2>
    <ul>
      <li>HTML5</li>
      <li>CSS 3</li>
      <li>活动沙龙</li>
    </ul>
  </div>
  <div class="center-content">
    <h2>文章列表</h2>
    <ul>
      <li>文本阴影</li>
      <li>色彩模式</li>
      <li>多重背景</li>
      <li>边框圆角</li>
      <li>新型盒布局</li>
      <li>盒子阴影</li>
    </ul>
  </div>
  <div class="right-aside">
    <h2>工具</h2>
    <ul>
      <li>天气预报</li>
      <li>货币汇率</li>
    </ul>
  </div>
</div>
</body>
</html>
```

运行结果如图 4-5 所示。



图 4-5　具有空间弹性的菜单栏

代码分析：在示例 4-5 中，设置 container 类的盒元素的宽度为 100%，即浏览器窗口的宽度，会随着窗口宽度的变化而变化。把菜单栏的 box-flex 属性设置为 1，使其具有空间弹性以分配盒元素的剩余空间。如图 4-5 所示，菜单栏由灰线开始填充了盒元素的剩余空间，菜单栏本身的宽度变大了。在本示例中，当窗口的宽度改变时，菜单栏的宽度也会跟着变化。

3. 多个子元素的弹性空间

当盒元素内部的多个子元素都定义 box-flex 属性时，子元素的空间弹性是相对的。浏览器将会把各个子元素的 box-flex 属性值相加得到一个总值，然后根据各自的值占总值的比例来分配盒元素的剩余空间。

对于示例 4-5 中的文章列表栏目也设置弹性，值为 2。调整样式表如下。

【示例 4-6】 多个子元素的弹性空间分配。

```
<style type="text/css">
.container {
    width:100%;                    /* 设置盒元素的宽度为 100% */
    background-color:#CCC;
    display:-webkit-box;
    display:-moz-box;
    display:box;
}
.container div {
    color:#FFF;
    font-size:12px;
    padding:10px;
    line-height:20px;
    width:100px;                   /* 设置三个栏目的固定宽度 100px */
}
.container div ul {
    margin:0;
    padding-left:20px;
}
.container .left-aside {
    background-color:#F63;
    /* 设置菜单栏具有空间弹性 */
    -webkit-box-flex:1;            /* 兼容 webkit 内核 */
    -moz-box-flex:1;               /* 兼容 gecko 内核 */
    box-flex:1;                    /* 标准用法 */
}
.container .center-content {
    background-color:#390;
    /* 设置文章列表栏有空间弹性 */
    -webkit-box-flex:2;            /* 兼容 webkit 内核 */
    -moz-box-flex:2;               /* 兼容 gecko 内核 */
    box-flex:2;                    /* 标准用法 */
}
.container .right-aside {
    background-color:#039;
}
</style>
```

运行结果如图 4-6 所示。



图 4-6　多个子元素的弹性空间分配

代码分析：在示例 4-6 中，菜单栏的 box-flex 属性值为 1，文章列表栏的 box-flex 属性值为 2。在分配剩余空间的时候，菜单栏仅分配 1/3 的剩余空间，文章列表则分配 2/3 的剩余空间。如图 4-6 所示，各栏目中灰线右边的部分为弹性分配的空间部分。

4.1.6　元素的对其方式——box-pack 和 box-align 属性

CSS 3 新增的 box-pack 属性和 box-align 属性，分别用于定义盒元素内部水平对齐方式和垂直对齐方式。这种对齐方式，对盒元素内部的文字、图像及子元素都是有效的。基于 webkit 内核的替代私有属性是-webkit-box-pack 和-webkit-box-align，基于 gecko 内核的替代私有属性是-moz-box-pack 和-moz-box-align。

1. 参数说明

box-pack 属性可设置子元素在水平方向上的对齐方式，其语法如下：

```
box-pack : start | end | center | justify;
```

取值说明如下。

- start：默认值，表示所有的子元素都显示在盒元素的左侧，额外的空间显示在盒元素右侧。
- end：表示所有的子元素都显示在盒元素的右侧，额外的空间显示在盒元素左侧。
- center：表示所有的子元素居中显示，额外的空间平均分配在两侧。
- justify：表示所有的子元素散开显示，额外的空间在子元素之间平均分配，在第一个子元素之前和最后一个子元素之后不分配空间。

box-align 属性可设置子元素在垂直方向上的对齐方式，其语法如下：

```
box-align : start | end | center | baseline | stretch;
```

取值说明如下。

- start：表示所有的子元素都显示在盒元素的顶部，额外的空间显示在盒元素底部。
- end：表示所有的子元素都显示在盒元素的底部，额外的空间显示在盒元素顶部。
- center：表示所有的子元素垂直居中显示，额外的空间平均分配在盒元素的上下两侧。
- baseline：表示所有的子元素沿着基线显示。

- stretch：默认值，表示每个子元素的高度被拉伸到适合的盒元素高度。

提示：box-pack 属性和 box-align 属性仅在盒布局模式下使用。在传统的对齐方式中，有 text-align 属性和 vertical-align 属性分别定义元素内的水平方向对齐和垂直方向对齐，但不宜用于盒布局。

2. 自适应居中

在 CSS 2.0 时代，把元素布局在页面的正中央，是难以通过 CSS 样式表实现的，通常会借助 JavaScript 技术来实现，需要写大量代码，还会遇到兼容性问题。下面我们借助 box-pack 属性和 box-align 属性，把一个对话框布局在页面中央。

【示例 4-7】 自适应居中的登录框。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>自适应居中的登录框</title>
<style type="text/css">
body, html {
    margin:0;
    padding:0;
    height:100%;
}
#box {
    width:100%;
    height:100%;
    background:url(images/sky.jpg) no-repeat 0 0;
    background-size:100% 100%;
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
    /* 水平居中 */
    -webkit-box-pack:center;
    -moz-box-pack:center;
    box-pack:center;
    /* 垂直居中 */
    -webkit-box-align:center;
    -moz-box-align:center;
    box-align:center;
}
#box div {
    opacity:0.8;
}
</style>
<body>
<div id="box">
  <div><img src="images/login.png" /></div>
</div>
</body>
</html>
```

运行结果如图 4-7 所示。



图 4-7　自适应居中的登录框

代码分析：示例 4-7 中，直接使用登录图片，仅为了说明问题。设置 box-pack 属性值为 center，使得登录框水平居中；设置 box-align 属性值为 center，使得登录框垂直居中。几行 CSS 样式就把登录框布局到中央了。

3. 底部对齐

在 CSS 2.0 时代，把元素布局在页面的底部也是比较困难的。下面借助 box-pack 属性和 box-align 属性来实现。

【示例 4-8】 把图片布局在盒元素底部。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>把图片布局在盒元素底部</title>
<style type="text/css">
#box {
    width:500px;
    height:200px;
    border:1px solid #F90;
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
    /* 左边对齐 */
    -webkit-box-pack:start;
    -moz-box-pack:start;
    box-pack:start;
    /* 底部对齐 */
    -webkit-box-align:end;
    -moz-box-align:end;
    box-align:end;
}
#box div {
    padding:5px;
    border:1px solid #ccc;
```

```
    margin:1px;
}
#box div img {
    width:120px;
}
</style>
<body>
<div id="box">
  <div><img src="images/IL1.jpg" /></div>
  <div><img src="images/IL2.jpg" /></div>
  <div><img src="images/IL3.jpg" /></div>
  <div><img src="images/IL4.jpg" /></div>
</div>
</body>
</html>
```

运行结果如图 4-8 所示。

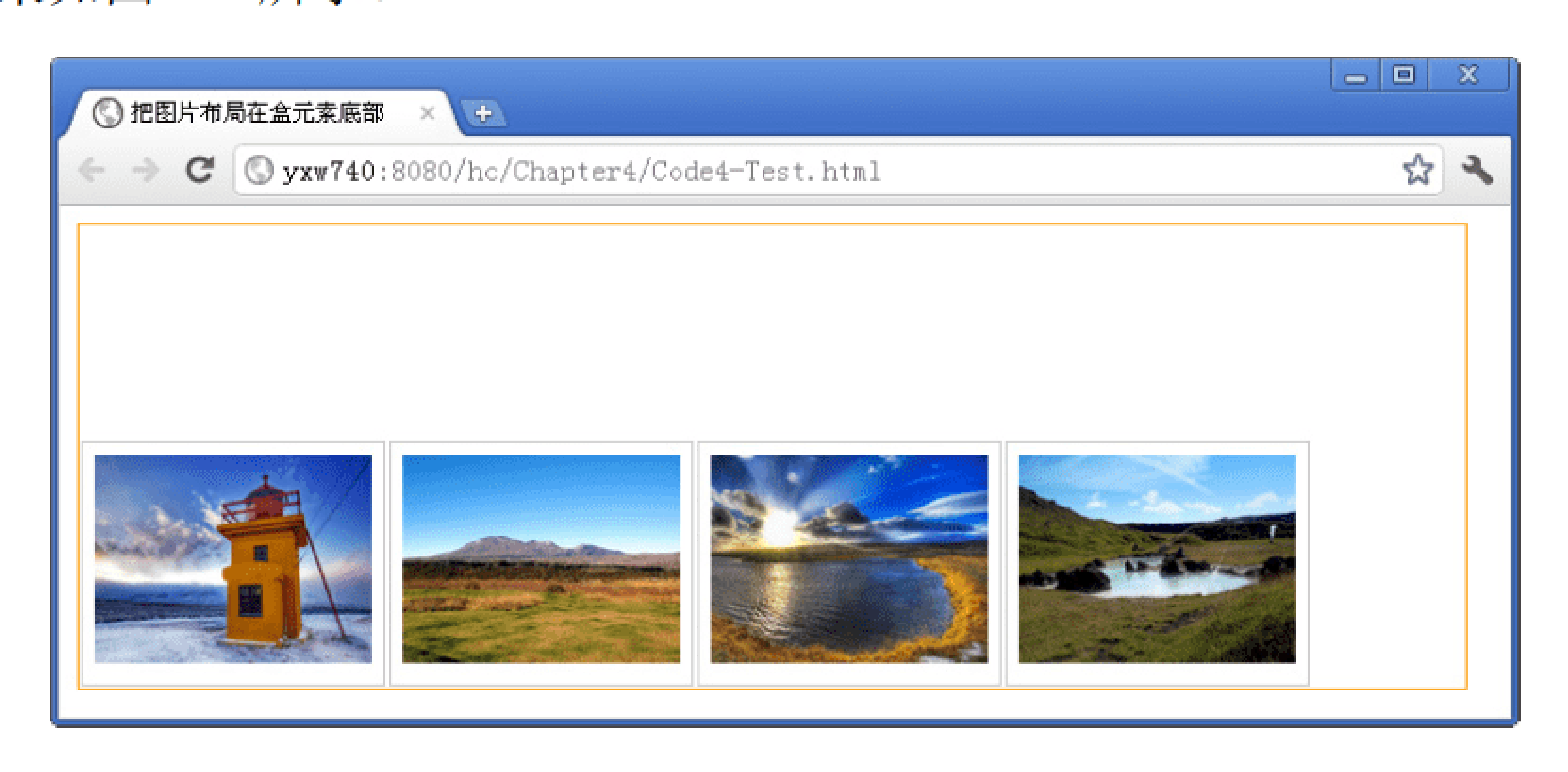


图 4-8　把图片布局在盒元素底部

代码分析：在示例 4-8 中，设置 box-pack 属性值为 start，使得子元素靠左边显示；设置 box-align 属性值为 end，使得子元素紧贴底部显示。如图 4-8 所示，额外的空间会出现在右侧和顶部。

提示： box-pack 属性和 box-align 属性的对齐方式的效果，还会受到 box-orient 属性和 box-direction 属性的影响。当 box-orient 属性设置为垂直方向时，box-pack 属性将控制垂直方向，box-align 属性将控制水平方向；当 box-direction 属性设置为反向时，对齐方式的属性值 start 和 end 将互换效果。

4.1.7　实验室：使用新型盒布局设计网页

CSS 3 发展的新型盒布局是一种全新的页面布局方式，不仅可以完全替代传统的浮动布局，而且布局出来的网页具有很强的适应性，具体的体现就是其空间弹性。下面就用刚刚学过的盒布局知识来布局网页。

1. 案例简介

本节要介绍的案例是一个典型的小型网页，从上至下包括页头、主体区域和页脚三个部分。其中主体部分从左至右分为三栏，分别是导航栏、文章栏、侧边栏。而文章内容栏

又可以从上至下分为：标题、内容、日期三栏。页面效果如图 4-9 所示，下面我们就逐步布局该网页。



图 4-9　使用新型盒布局设计网页

2. 设计网页元素

下面给出网页中的 HTML 标签代码和基本的样式表设置。

【示例 4-9】　使用新型盒布局设计网页。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>使用新型盒布局设计网页</title>
<style type="text/css">
h3 {
    padding:15px;
    color:#FFFFFF;
    margin:0px;
}
a {color:#fff;}
</style>
</head>
<body>
<div id="area">
  <header id="header">页头：使用新型盒布局设计网页 </header>
  <div id="container">
    <nav>
      <h3>导航栏</h3>
      <a href="#">盒布局</a> <a href="#">布局方向</a> <a href="#">布局顺序</a>
      </nav>
    <article>
      <header>标题：开启盒布局 </header>
      <p>内容：盒布局是 CSS 3 发展的新型布局方式，它比传统的浮动布局方式更加完善、更加
      灵活，而使用方法却极为简单。
        开启盒布局后，文档就会按照盒布局默认的方式，来布局子元素。</p>
      <footer> 日期：2011-10 </footer>
```

```
    </article>
    <aside>
      <h3>侧边栏</h3>
      <p>侧边栏内容</p>
    </aside>
  </div>
  <footer id="footer"> 页脚：2011&copy; </footer>
</div>
</body>
</html>
```

下面一步一步地给各个功能块编写样式表，并追加到示例 4-9 中。

3. 从body元素开始弹性布局

把 body 元素定义为盒布局，并设置子元素水平居中。

```
body, html {
    margin:0;
    padding:0;
    width:100%;
    height:100%;
    font-family:Arial, Helvetica, sans-serif;
}
body {
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
    /* 设置盒子内元素水平居中 */
    -webkit-box-pack:center;
    -moz-box-pack:center;
    box-pack:center;
}
```

4. 定义网页区域

设置网页区域的空间弹性以占满整个区域。使用盒布局，并设置垂直方向布局。

```
#area {
    height:100%;
    max-width:950px;              /* 最大宽度  */
    min-width:600px;              /* 最小宽度 */
    /* 定义空间弹性，使其充满页面空间，但宽度受 max-width 和 min-width 限制 */
    -webkit-box-flex:1;
    -moz-box-flex:1;
    box-flex:1;
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
    /* 垂直布局，实现竖直方向的三栏布局 */
    -webkit-box-orient:vertical;
    -moz-box-orient:vertical;
    box-orient:vertical;
}
```

5. 设置页头、主体区域和页脚三个部分

设置页头、主体区域和页脚三个部分。其中，主体区域部分定义空间弹性，并使用盒布局及默认的水平方向。其他部分为常规样式表。

```
#area header {
    background-color:#ff6600;
    text-align:center;
    line-height:35px;
    color:#FFFFFF;
    font-size:24px;
    font-weight:bold;
}
#area #header {
    padding:15px;
}
#area #container {
    /* 定义空间弹，使其随空间伸缩尺寸 */
    -webkit-box-flex:1;
    -moz-box-flex:1;
    box-flex:1;
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
}
#area footer {
    background-color:#f47D31;
    text-align:center;
    line-height:20px;
    color:#FFFFFF;
}
#area #footer {
    padding:10px;
}
```

6. 设置主体区域中的导航栏、文章栏、侧边栏三个部分

设置主体区域中的导航栏、文章栏、侧边栏三个部分。其中，文章栏定义空间弹性，并使用盒布局和垂直方向布局。其他部分为常规样式表。

```
#area #container nav {
    width:170px;
    background-color:#999;
}
#area #container article {
    padding:10px;
    /* 定义空间弹，使其随空间伸缩尺寸 */
    -webkit-box-flex:1;
    -moz-box-flex:1;
    box-flex:1;
    /* 开启盒布局 */
    display:-webkit-box;
    display:-moz-box;
    display:box;
    /* 布局方向设置为竖直方向 */
    -webkit-box-orient:vertical;
```

```
    -moz-box-orient:vertical;
    box-orient:vertical;
}
#area #container aside {
    width:170px;
    background-color:#999;
}
```

7. 分别设置导航栏、文章栏、侧边栏的详细部分

分别设置导航栏、文章栏、侧边栏的详细部分。其中文章栏中的文章内容部分设置为空间弹性。其他部分为常规样式表。

```
/* 左侧导航 */
#area #container nav a:link, #area #container nav a:visited {
    display:block;
    border-bottom:3px solid #fff;
    padding:10px;
    text-decoration:none;
    font-weight:bold;
    margin:5px;
}
#area #container nav a:hover {
    color:#FFFFFF;
    background-color:#f47D31;
}
/* 中间内容 */
#area #container article p {
    -webkit-box-flex:1;
    -moz-box-flex:1;
    box-flex:1;
}
/* 侧边栏 */
#area #container aside p {
    padding:15px;
    font-weight:bold;
    font-style:italic;
    color:#FFF;
}
```

至此，一个典型的小型网页设计完成。该网页在改变窗口大小的情况下，能继续保持页面总体格局的完整。

4.2　增强的盒模型

盒模型是网页设计中最基本、最重要的模型。CSS 3 新增的与盒模型有关的属性如盒子阴影、盒子尺寸和溢出处理等，为前端设计师带来更多的便利及人性化设计。

4.2.1　盒子阴影——box-shadow 属性

CSS 3 新增的 box-shadow 属性，可以定义元素的阴影效果。关于该属性，设计师们尤其喜欢。到目前为止，已经获得更多浏览器的支持和更加广泛的使用。基于 webkit 内核的

替代私有属性是-webkit-box-shadow，基于 gecko 内核的替代私有属性是-moz-box-shadow。

1. 参数说明

box-shadow 属性为盒元素添加一个或多个阴影。其语法如下：

```
box-shadow : none | [inset]? [<length>] {2,4} [<color>]?;
```

取值说明如下。

- none：默认值，表示没有阴影。
- inset：可选值，表示设置阴影的类型为内阴影，默认为外阴影。
- <length>：是由浮点数字和单位标识符组成的长度值，可取负值。4 个 length 分别表示阴影的水平偏移、垂直偏移、模糊距离和阴影大小，其中水平偏移和垂直偏移是必需的值，模糊半径和阴影大小可选。
- <color>：可选，表示阴影的颜色。

完整的阴影属性值包含 6 个参数值：阴影类型、水平偏移长度、垂直偏移长度、模糊半径、阴影大小和阴影颜色，其中水平偏移长度和垂直偏移长度是必需的，其他的都可以有选择地省略。

提示： 盒子阴影与文本阴影看起来很相像，但是它们的语法也是不同的，而且盒子阴影应用于页面元素，文本阴影仅应用于文字。

2. 盒子的阴影效果

下面通过示例来全面了解 box-shadow 属性的使用方法。该示例是为橘黄色的盒元素设置阴影。其中阴影在水平和垂直方向上的距离均为 5px，模糊作用距离为 5px，阴影颜色为深灰色。

【示例 4-10】 给盒元素添加阴影效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>盒子阴影</title>
<style type="text/css">
div {
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow:5px 5px 5px #333;       /* 兼容 webkit 内核 */
    -moz-box-shadow:5px 5px 5px #333;          /* 兼容 gecko 内核 */
    box-shadow:5px 5px 5px #333;               /* 标准用法 */
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 4-10 所示。



图 4-10　盒元素的阴影效果

代码分析：在示例 4-10 中，我们为盒子的阴影属性设置了水平偏移值、垂直偏移值、模糊距离和阴影颜色。阴影类型是默认的外部阴影，如图 4-10 所示。

如果水平偏移值和垂直偏移值为负数，表示阴影向左或向上偏移。调整样式表如下：

```
<style type="text/css">
div {
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow:-5px -5px 5px #00f;      /* 兼容 webkit 内核 */
    -moz-box-shadow:-5px -5px 5px #00f;         /* 兼容 gecko 内核 */
    box-shadow:-5px -5px 5px #00f;              /* 标准用法 */
}
</style>
```

运行结果如图 4-11 所示。



图 4-11　盒元素的左上方阴影效果

box-shadow 属性还可以同时使用两种及两种以上的阴影。调整样式表如下：

```
<style type="text/css">
div {
    margin:20px auto;
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow:-5px -5px 5px 0 #00f,
                5px 5px 5px 0 #333;
    -moz-box-shadow:-5px -5px 5px 0 #00f,
            5px 5px 5px 0 #333;
    box-shadow:-5px -5px 5px 0 #00f,
        5px 5px 5px 0 #333;
}
</style>
```

运行结果如图 4-12 所示。



图 4-12　同时应用两种阴影的效果

3. 盒子的描边效果

用 box-shadow 属性可以给盒子设置丰富的描边，这种描边效果可以替代单调的边框。实现描边的方法是把水平偏移值和垂直偏移值设置为 0，仅设置模糊半径、阴影大小和阴影颜色。

【示例 4-11】 设计盒子的描边效果。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>盒子描边效果</title>
<style type="text/css">
div {
    margin:20px auto;
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow:0 0 5px 5px #333;
    -moz-box-shadow:0 0 5px 5px #333;
    box-shadow:0 0 5px 5px #333;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 4-13 所示。



图 4-13　盒子的描边效果

代码分析：示例 4-11 中，仅设置了模糊半径、阴影大小和颜色，就可以实现如图 4-13 所示的描边效果。如果不设置模糊半径，则描边效果就等同于边框效果了。调整样式表如下：

```
<style type="text/css">
div {
    margin:20px auto;
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow:0 0 0 5px #333;
    -moz-box-shadow:0 0 0 5px #333;
    box-shadow:0 0 0 5px #333;
}
</style>
```

运行结果如图 4-14 所示。



图 4-14　没有模糊半径的描边效果

4. 盒子的内阴影

box-shadow 属性中添加值 inset，即可实现向内的阴影。

【示例 4-12】 设计盒子的内阴影。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>盒子的内阴影</title>
<style type="text/css">
div {
    width:200px;
    height:100px;
    background-color:#f90;
    -webkit-box-shadow: inset 5px 5px 5px #333;    /* 兼容 webkit 内核 */
    -moz-box-shadow: inset 5px 5px 5px #333;       /* 兼容 gecko 内核 */
    box-shadow: inset 5px 5px 5px #333;            /* 标准用法 */
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 4-15 所示。



图 4-15　盒子的内阴影效果

代码分析：示例 4-12 中，在 box-shadow 属性中增加属性值 inset，即实现了内阴影。

4.2.2　盒子尺寸的计算方法——box-sizing 属性

对于前端工程师和设计人员来说，应该都有过这样的经历：当为一个盒元素同时设置 border、padding 和 width 或 height 属性时，在不同的浏览器下，会有不同的尺寸。特别是在 IE 浏览器中，width 和 height 是包含 border 和 padding 的，标准的 width 和 height 是不包含 border 和 padding 的。为此，要写大量的 hack，以满足不同浏览器的需要。

CSS 3 对盒模型进行了改善，新增的 box-sizing 属性，可用于定义 width 和 height 的计算方法，可自由定义是否包含 border 和 padding。

1. 参数说明

box-sizing 属性定义盒元素尺寸的计算方法。其语法如下：

```
box-sizing : content-box | padding-box | border-box | inherit ;
```

取值说明如下。

- ❑ content-box：默认值，计算方法为 width/height=content，表示指定的宽度和高度仅限内容区域，边框和内边距的宽度不包含在内。
- ❑ padding-box：计算方法为 width/height=content+padding，表示指定的宽度和高度包含内边距和内容区域，边框宽度不包含在内。
- ❑ border-box：计算方法为 width/height=content+padding+border，表示指定的宽度和高度包含边框、内边距和内容区域。
- ❑ inherit：表示继承父元素中 box-sizing 属性的值。

2. 示例介绍

下面的示例中，把 div 标签默认设置为宽 200px，高 80px，边框宽 10px，内边距宽 10px。然后分别设置不同的属性 box-sizing 的值，以观察各个属性值的区别。

【示例 4-13】 盒子尺寸的计算方法。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>盒子尺寸的计算方法</title>
```

```
<style type="text/css">
div {
    margin:5px;
    width:200px;                /* 宽度为 200px */
    height:80px;                /* 高度为 80px */
    background-color:#fe0;
    border:10px solid #f90;     /* 边框宽度为 10px */
    padding:10px;               /* 内边距宽度为 10px */
    font-weight:bold;
    font-size:18px;
    line-height:40px;
}
/* 属性值 border-box */
.s1 {
    box-sizing:border-box;
    -webkit-box-sizing:border-box;
    -moz-box-sizing:border-box;
}
/* 属性值 padding-box */
.s2 {
    box-sizing:padding-box;
    -webkit-box-sizing:padding-box;
    -moz-box-sizing:padding-box;
}
/* 属性值 content-box */
.s3 {
    box-sizing:content-box;
    -webkit-box-sizing:content-box;
    -moz-box-sizing:content-box;
}
</style>
</head>
<body>
<div class="s1">border-box</div>
<div class="s2">padding-box</div>
<div class="s3">content-box</div>
</body>
</html>
```

运行结果如图 4-16、图 4-17 所示。

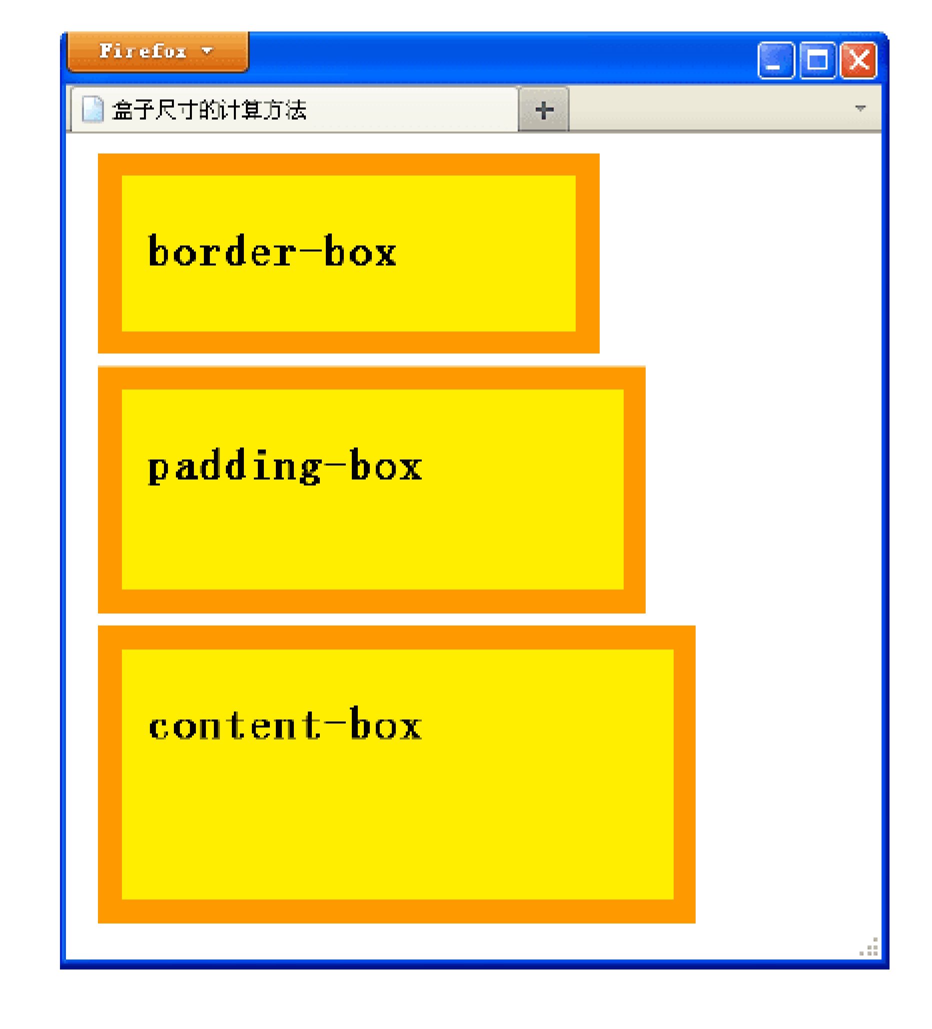


图 4-16　Firefox 中预览的效果

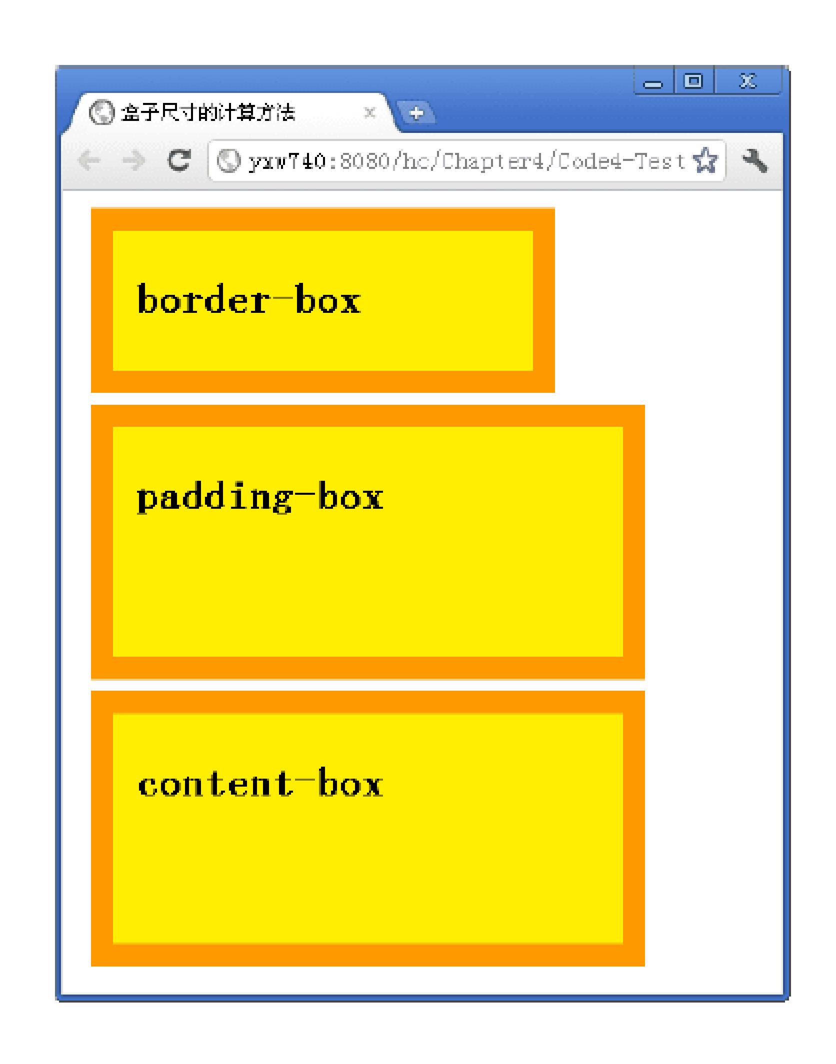


图 4-17　chrome 中预览的效果

代码分析：在示例 4-13 中，由于尺寸的计算方式不同，元素的总尺寸也不相同。由于 webkit 内核的浏览器不支持属性值 padding-box，故表现出默认的 content-box 效果。

4.2.3　盒子溢出内容处理——overflow-x 和 overflow-y 属性

在 CSS 2.1 规范中，就已经有处理溢出的 overflow 属性，该属性定义当盒子的内容超出盒子边界时的处理方法。CSS 3 新增的 overflow-x 和 overflow-y 属性，是对 overflow 属性的补充，分别表示水平方向上的溢出处理和垂直方向上的溢出处理。

1. 参数说明

overflow-x 和 overflow-y 属性的语法如下：

```
overflow-x : visible | auto | hidden | scroll | no-display | no-content;
overflow-y : visible | auto | hidden | scroll | no-display | no-content;
```

取值说明：

- visible：默认值，盒子溢出时，不裁剪溢出的内容，超出盒子边界的部分将显示在盒元素之外。
- auto：盒子溢出时，显示滚动条。
- hidden：盒子溢出时，溢出的内容将被裁剪，并且不提供滚动条。
- scroll：始终显示滚动条。
- no-display：当盒子溢出时，不显示元素。该属性值是新增的。
- no-content：当盒子溢出时，不显示内容。该属性值是新增的。

2. 示例介绍

下面通过一个示例来介绍溢出处理的各个属性值的应用效果。

【示例 4-14】 盒子溢出内容处理。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>盒子溢出内容处理</title>
<style type="text/css">
div {
    margin:10px;
    width:200px;
    height:80px;
    padding:10px;
    border:1px solid #f90;
    float:left;
}
#box1 {
    overflow-x:scroll;     /* 水平方向属性值为 scroll */
    overflow-y:scroll;     /* 垂直方向属性值为 scroll */
}
#box2 {
    overflow-x:auto;       /* 水平方向属性值为 auto */
    overflow-y:auto;       /* 垂直方向属性值为 auto */
```

```
}
#box3 {
    overflow-x:hidden;      /* 水平方向属性值为 hidden */
    overflow-y:hidden;      /* 垂直方向属性值为 hidden */
}
#box4 {
    overflow-x:visible;     /* 水平方向属性值为 visible */
    overflow-y:visible;     /* 垂直方向属性值为 visible */
} </style>
</head>
<body>
<div id="box1">盒模型是网页设计中最基本、最重要的模型。。。</div>
<div id="box2">盒模型是网页设计中最基本、最重要的模型。。。</div>
<div id="box3">盒模型是网页设计中最基本、最重要的模型。。。</div>
<div id="box4">盒模型是网页设计中最基本、最重要的模型。。。</div>
</body>
</html>
```

运行结果如图 4-18 所示。



图 4-18　盒子溢出内容的处理效果

代码分析：在示例 4-14 中，分别为每个 div 元素设置不同的溢出处理方式。如图 4-18 所示，溢出属性值为 scroll 时，水平方向和垂直方向均显示滚动条；溢出属性值为 auto 时，只有在需要滚动条的时候，才会显示滚动条；溢出属性值为 hidden 时，会裁剪一部分溢出的内容；溢出属性值为 visible 时，溢出的内容会显示在盒子边界的外面。

4.2.4　实验室：设计网站服务条款页面

当一些网站提供一些服务时，都会有相应的服务条款。服务条款的页面比较简单，但可以实现不同的风格。下面将使用前面学过的样式表属性来修饰这个页面。

1. 案例简介

本节介绍的案例只有一个主体区域，我们将把该主体区域设计成凸起的效果，然后在

这个凸起的区域里放置内容。由于条款内容较长，还需要处理内容的溢出，页面效果如图 4-19 所示。下面就逐步设计该网页。



图 4-19　网站服务条款效果图

2. 设计网页元素

网页只有一个主体区域，该区域包括头部的标题、中间的条款区域和底部的确认按钮。

【示例 4-15】 设计网站服务条款页面。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>网站服务条款</title>
</head>
<body>
<div id="policy">
  <header>网站服务条款</header>
  <section>
    <p> 用户必须单独承担发布内容的责任。用户对论坛服务的使用是根据所有适用于本论坛的国家法律、地方法律和国际法律标准的。</p>
    <p> 用户不得在本站论坛发布含有下列内容之一的信息:</p>
    <p>(1) 反对宪法所确定的基本原则; </p>
    <p>(2) 危害国家安全,泄露国家秘密,颠覆国家政权,破坏国家统一; </p>
    <p>(3) 损害国家荣誉和利益; </p>
    <p>(4) 煽动民族仇恨、民族歧视,破坏民族团结; </p>
    <p>(5) 破坏国家宗教政策,宣扬邪教和封建迷信; </p>
    <p>(6) 散布谣言,扰乱社会秩序,破坏社会稳定;</p>
    <p>(7) 散布淫秽、色情、赌博、暴力、凶杀、恐怖或者教唆犯罪; </p>
    <p>(8) 侮辱或者诽谤他人,侵害他人合法权益;</p>
    <p>(9) 连锁信件,金字塔方案及蛊惑性文章;</p>
    <p>(10) 任何涉及他人版权的资料的非法复制和传播; </p>
    <p>(11) 任何未经本站许可的商业性质的广告; </p>
```

```
    <p>(12) 含有法律、行政法规禁止的其他内容。</p>
  </section>
  <footer>
    <input type="button" value="同意" />
    <input type="button" value="不同意" />
  </footer>
</div>
</body>
</html>
```

下面，我们逐步设计样式表，并追加到示例 4-15 中。

3. 首先设计主体区域样式

设置主体区域页面居中；设置尺寸及其计算方式，以保证内部的条款区域的空间；设置背景颜色及向内的阴影，以实现主体区域的凸起效果。

```
<style type="text/css">
#policy {
    font-family:Arial, 宋体;
    margin:10px auto;                  /* 页面居中 */
    box-sizing:content-box;            /* 尺寸计算方式为 content-box */
    width:400px;                       /* 盒子宽 400px */
    padding:20px;                      /* 内边距为 20px */
    background-color:#e4e4e4;          /* 浅灰色背景 */
    box-shadow:inset 0 0 15px 5px #bbb;   /* 向内的阴影 */
}
</style>
```

4. 设计主体区域内部的样式

分别设计主体区域中的头部标题、中间的条款和底部的按钮。其中，中间条款的尺寸包含到边框，水平方向不设置滚动条，垂直方向设置滚动条。底部按钮也增加了阴影效果。

```
<style type="text/css">
#policy header {
    font-size:24px;
    font-weight:bold;
    line-height:25px;
}
#policy section {
    background-color:#fff;
    font-size:12px;
    line-height:25px;
    box-sizing:border-box;      /* 尺寸计算方式为 border-box */
    width:400px;                /* 盒子宽 400px */
    height:200px;               /* 盒子高 200px */
    padding:10px;               /* 内边距为 10px */
    border:1px solid #CCC;
    overflow-x:hidden;          /* 水平方向不设置滚动条 */
    overflow-y:scroll;          /* 垂直方向设置滚动条 */
}
#policy footer {
    text-align:center;
    padding-top:5px;
}
```

```
#policy footer input {
    border:1px solid #666;
    box-shadow:2px 2px 1px #BBB;    /* 按钮阴影 */
}
</style>
```

至此，完成了服务条款页面的设计，即可以展现如图 4-19 所示的页面效果。该页面没有借助任何图片，全部使用 CSS 设计完成。

4.3　增强的用户界面设计

在界面设计方面，为了增强用户体验，设计师们会想尽办法来实现心目中的页面效果，也因此徒增很多工作量。CSS 3 在用户界面的设计方面有很大的改进，可以允许改变元素尺寸、定义外轮廓线、改变焦点导航顺序、让元素变身，以及给元素添加内容等。这些功能的改进，使得设计师设计出的页面具有更加强大的用户体验。

4.3.1　允许用户改变尺寸——resize 属性

如果你在使用 Firefox 或 Chrome，那么你肯定注意到了 textarea 标签元素的右下角默认有个小的手柄，它可以让你调整元素的大小。CSS 3 新增的 resize 属性，可以对其他元素也应用同样的效果。

1. 参数说明

resize 属性定义一个元素是否允许用户调整大小。其语法如下：

```
resize : none | both | horizontal | vertical | inherit ;
```

取值说明如下。

- none：默认值，表示用户不能调整元素。
- both：表示用户可以调整元素的宽度和高度。
- horizontal：表示用户可以调整元素的宽度，但不能调整元素的高度。
- vertical：表示用户可以调整元素的高度，但不能调整元素的宽度。
- inherit：表示继承父元素。

提示： resize 属性需要和溢出处理属性 overflow 或 overflow-x 或 overflow-y 一起使用，才能把元素定义成可以调整大小的，且溢出属性值不能为 visible。

2. 示例介绍

下面看一下 resize 属性的如何使用及其使用效果。

【示例 4-16】 可以调整大小的 div 元素。

```
<!DOCTYPE HTML>
<html>
<head>
```

```
<meta charset="utf-8">
<title>可以调整大小的 div 元素</title>
<style type="text/css">
div {
    width:100px;
    height:80px;
    max-width:600px;          /* 设置最大宽度限制 */
    max-height:400px;         /* 设置最大高度限制 */
    padding:10px;
    border:1px solid #f90;
    resize:both;              /* 设置元素的宽度和高度均可调整 */
    overflow:auto;            /* 设置溢出属性值为 auto */
}
</style>
</head>
<body>
<div> 如果你在使用 Firefox 或 Chrome，那么你肯定注意到了 textarea 标签元素的右下角默认有个小的手柄，它可以让你调整它们的大小。
```

CSS 3 新增的 resize 属性，可以对其他元素也应用同样的效果。resize 属性定义一个元素是否允许用户调整大小。</div>

```
</body>
</html>
```

运行结果如图 4-20 所示。

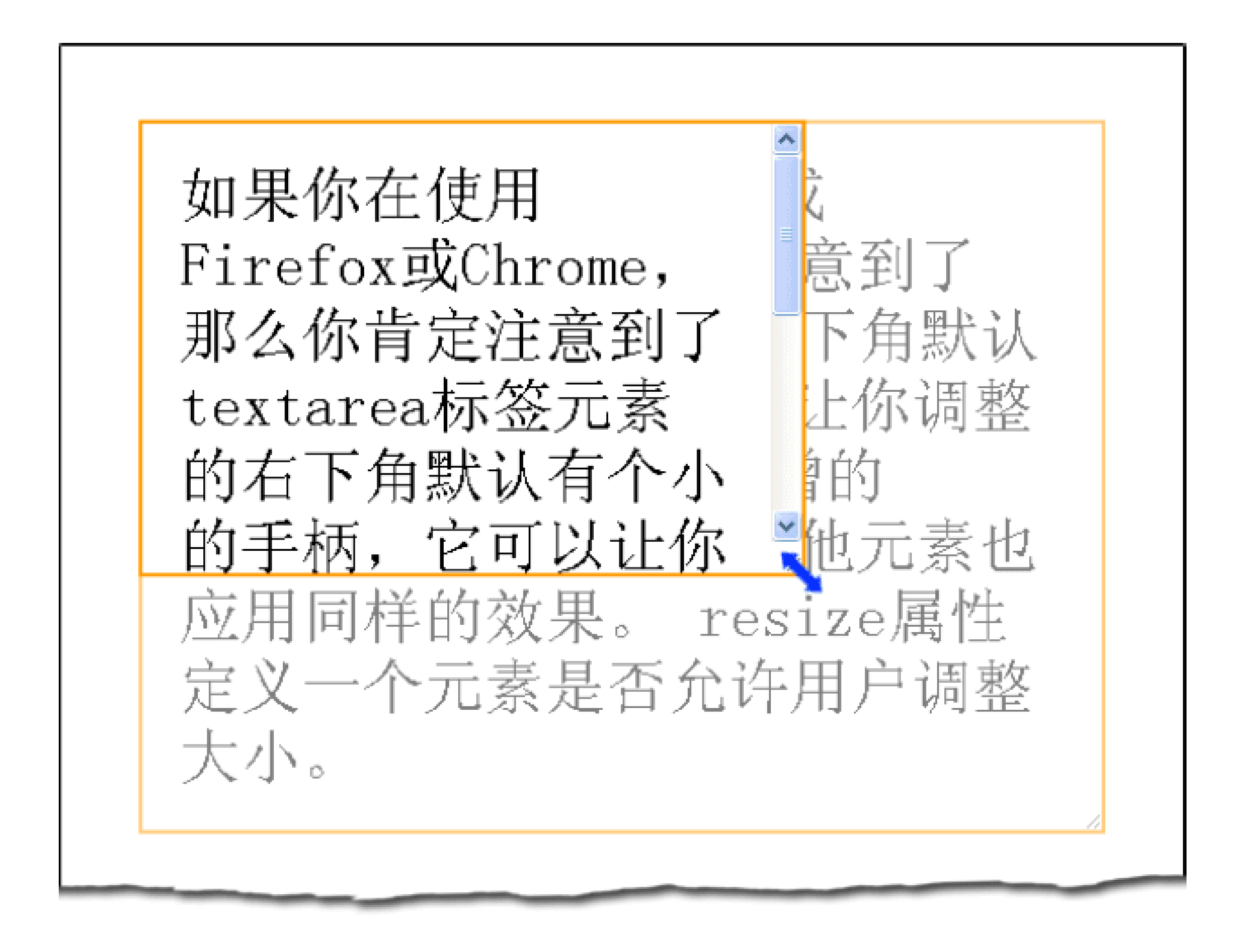


图 4-20　可以自由调整的 div 元素

代码分析：在示例 4-16 中，设置 resize 属性值为 both，并且设置了 overflow 属性为 auto，这样在运行出来的页面里就可以调整该元素的大小。示例中还设置了最大宽度和最大高度，在用户调整元素的大小时，会限制在最大尺寸范围内。如图 4-20 所示为元素调整大小前后的变化。

4.3.2　定义外轮廓线——outline 属性

outline 属性可以定义一个元素的外轮廓线，以突出显示该元素。外轮廓线看起来很像

元素边框，而且语法也与边框非常类似，但是外轮廓线不占用元素的尺寸。outline 属性属性在 CSS 2.1 中已经定义，但没有获得浏览器支持。CSS 3 完善并增强了该属性，并获得了各主流浏览器的支持。

1. 参数说明

outline 属性用来在元素周围定义轮廓线，其语法如下：

```
outline : [outline-width] || [outline-style] || [outline-color] |inherit ;
```

取值说明如下。

- <outline-width>：定义元素轮廓的宽度。
- <outline-style>：定义元素轮廓的样式风格。
- <outline-color>：定义元素轮廓的颜色。
- inherit：表示继承父元素的轮廓样式。

outline 属性与 border 属性有很多相似的地方，但也有很大的不同。outline 属性定义的外轮廓线总是封闭的、完全闭合的；外轮廓线也可能不是矩形，如果元素的 display 属性值为 inline，外轮廓就可能变得不规则。

outline 属性是一个复合的属性。它包含了 4 个子属性：outline-width 属性、outline-style 属性、outline-color 属性和 outline-offset 属性。

2. 轮廓的宽度属性outline-width

outline-width 属性，用于定义元素轮廓的宽度。语法如下：

```
outline-width : thin | medium | thick | <length> | inherit ;
```

取值说明如下。

- thin：定义较细的轮廓宽度。
- medium：为默认值，定义中等的轮廓宽度。
- thick：定义较粗的轮廓宽度。
- <length>：定义轮廓的宽度值，宽度值包含长度单位，不允许为负值。
- inherit：表示继承父元素。

3. 轮廓的风格属性outline-style

outline-style 属性，用于定义元素轮廓的风格样式。语法如下：

```
outline-style : none | dotted | dashed | solid | double | groove | ridge
| inset | outset | inherit ;
```

取值说明如下。

- none：定义没有轮廓。
- dotted：定义轮廓为点状。
- dashed：定义轮廓为虚线。
- solid：定义轮廓为实线。
- double：定义轮廓为双线条，双线的宽度等于 outline-width 属性的值。

- groove：定义轮廓为 3D 凹槽，显示效果取决于 outline-color 属性的值。
- ridge：定义轮廓为 3D 凸槽，显示效果取决于 outline-color 属性的值。
- inset：定义轮廓为 3D 凹边，显示效果取决于 outline-color 属性的值。
- outset：定义轮廓为 3D 凸边，显示效果取决于 outline-color 属性的值。
- inherit：表示继承父元素。

4. 轮廓的颜色属性outline-color

outline-color 属性，用于定义元素轮廓的颜色。语法如下：

```
outline-color : <color> | invert | inherit ;
```

取值说明如下。

- <color>：表示颜色值。CSS 中可使用的任何颜色，也可以是半透明颜色。
- invert：为默认值，执行颜色反转，以保证轮廓在任何背景下都是可见的。
- inherit：表示继承父元素。

5. 轮廓的偏离属性outline-offset

outline-offset：用于定义外轮廓与元素边界的距离。语法如下：

```
outline-offset : <length> | inherit ;
```

取值说明如下。

- <length>：表示偏离距离的长度值，长度值包含长度单位，可以为负值。
- inherit：表示继承父元素。

提示：在复合属性 outline 的语法中，没有包含 outline-offset 子属性，因为这样会造成长度值指定不明确，无法正确解析。

6. 示例介绍

下面我们用 outline 属性来设置表单。该示例中把整个登录区域、文本框和按钮，均设置 outline 属性，以展现该属性的用途。

【示例 4-17】 使用外轮廓渲染登录表单。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>使用外轮廓渲染登录表单</title>
<style type="text/css">
#login {
    margin:20px auto;
    width:300px;
    border:1px solid #f90;
    padding:20px;
    line-height:22px;
    outline:2px solid #ccc;                /* 设置外轮廓 */
    background-color:#ffff99;
    font-size:18px;
}
```

```
#login h1 {
    font-size:18px;
    margin:0;
    padding:5px;
    border-bottom:1px solid #fc6;
    margin-bottom:10px;
}
#login label {
    display:block;
    width:100px;
    float:left;
    text-align:right;
    clear:left;
    margin-top:15px;
}
#login input {
    float:left;
    width:150px;
    margin-top:15px;
    line-height:22px;
    height:24px;
    border:1px solid #7f9db9;
}
#login input:focus {
    outline:4px solid #fc6;                /* 设置外轮廓 */
}
#login div {
    clear:both;
    padding-left:100px;
    padding-top:20px;
    font-size:12px;
}
#login div button {
    width:80px;
    font-size:14px;
    line-height:22px;
    background-image: -moz-linear-gradient(top, #ffffcc, #ffcc99);
                                           /* 渐变 */
    background-image: -webkit-gradient(linear, left top, left bottom,
    from(#ffffcc), to(#ffcc99));           /* 渐变 */
    border:1px solid #f90;
}
#login div button:hover {
    outline:2px solid #fc6;                /* 设置外轮廓 */
}
</style>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
  <div id="login">
    <h1>用户登录</h1>
    <label for="UserName">用户名: </label>
    <input type="text" name="UserName" id="UserName">
    <label for="Password">密码: </label>
    <input type="password" name="Password" id="Password">
    <div><button>登录</button><a href="#">忘记密码? </a> </div>
  </div>
</form>
</body>
</html>
```

运行结果如图 4-21 所示。

图 4-21　使用外轮廓渲染登录表单

代码分析：在示例 4-17 中，均使用复合属性 outline 设置轮廓。整个登录区域设置了外轮廓线，以丰富边框效果；文本框获取焦点时，设置外轮廓线，以突出获取焦点的文本框；鼠标经过登录按钮时，也设置外轮廓线突出显示，效果如图 4-21 所示。

如果设置 outline-offset 属性，则外轮廓线就不会紧贴元素显示，看起来也更加大方。调整文本框焦点样式如下：

```
<style type="text/css">
…
#login input:focus {
    outline:4px solid #fc6;             /* 设置外轮廓 */
    outline-offset:5px;                 /* 设置外轮廓与元素边界的距离 */
}
…
</style>
```

运行结果如图 4-22 所示。

图 4-22　轮廓远离文本框边界

如果设置 outline-offset 属性为负值时，则轮廓线就会显示在元素的内部。调整文本框焦点样式如下：

```
<style type="text/css">
…
#login input:focus {
    outline:4px solid #fc6;                /* 设置外轮廓 */
    outline-offset:-7px;                   /* 设置外轮廓与元素边界的距离 */
}
…
</style>
```

运行结果如图 4-23 所示。



图 4-23　轮廓在文本框内部

4.3.3　伪装的元素——appearance 属性

你是否曾经把链接伪装成按钮，或者在伪装的按钮上输入字符？CSS 新增的 appearance 属性，可以方便地把元素伪装成其他类型的元素，给界面设计带来极大的灵活性。基于 webkit 内核的替代私有属性是-webkit-appearance，基于 gecko 内核的替代私有属性是-moz-appearance。

1. 参数说明

appearance 属性用于定义一个元素看起来像其他元素。语法如下：

```
appearance : normal | icon | window | button | menu | field ;
```

取值说明如下。

- ❑ normal：正常地修饰元素。
- ❑ icon：把元素修饰得像一个图标。
- ❑ window：把元素修饰得像一个视窗。
- ❑ button：把元素修饰得像一个按钮。
- ❑ menu：把元素修饰得像菜单。

- field：把元素修饰得像一个输入框。

提示： 需要说明的是，使用 appearance 属性定义的元素，仍然保留元素的功能，仅在外观上做了改变。由于受到元素本身功能的局限，不是每一个元素都可以任意被修饰，但是恰当地修饰大部分是可行的。

2. 示例介绍

下面的示例使用 appearance 属性，把页面元素均伪装成按钮。

【示例 4-18】 伪装的按钮。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>伪装的按钮</title>
<style type="text/css">
div, a, input {
    -webkit-appearance:button;          /* 修饰为按钮风格 */
    -moz-appearance:button;             /* 修饰为按钮风格 */
    appearance:button;                  /* 修饰为按钮风格 */
}
#nav {
    width:240px;
    padding:10px;
    height:130px;
    font-size:14px;
}
#nav a {
    font-size:12px;
    padding:0 10px;
    line-height:22px;
    text-decoration:none;
    color:#00F;
}
</style>
</head>
<body>
<div id="nav">
  <input type="text" name="key" value="关键词">
  <a href="#">搜索</a><br>
  热门关键词：<br>
<a href="#">CSS 3</a><a href="#">HTML5</a><a href="#">移动开发</a></div>
</body>
</html>
```

运行结果如图 4-24 所示。

代码分析：在示例 4-18 中，没有添加任何按钮元素，把标签 div、a 和 input 均修饰为按钮风格。在如图 4-24 所示的显示效果中，所有的链接被修饰成按钮风格，div 标签和输入框也是按钮风格，呈现出来的按钮都是伪装的。



图 4-24　轮廓在文本框内部

4.3.4　为元素添加内容——content 属性

如果要给元素插入内容，很少有人会想到使用 CSS 样式表来实现。在 CSS 中，可以使用 content 属性为元素添加内容，这已然替代了 JavaScript 的部分功能。

content 属性早在 CSS 2.1 的时候就被引入了，可以使用:before 及:after 伪元素生成内容，CSS 3 仍然引用了该属性，并且已经获得广泛的支持。

1. 参数说明

```
content : none | normal | <string> | counter(<counter>) | attr(<attribute>)
| url(<url>) | inherit ;
```

取值说明如下。

- ❑ none：如果有指定的添加内容，则设置为空。
- ❑ normal：默认值，不做任何指定或改动。
- ❑ <string>：指定添加的文本内容。
- ❑ counter（<counter>）：指定一个计数器作为添加内容。
- ❑ attr（<attribute>）：把选择的元素的属性值作为添加内容，<attribute>为元素的属性。
- ❑ url（<url>）：指定一个外部资源作为添加内容，如图像、音频、视频等，<url>为一个网络地址。
- ❑ inherit：表示继承父元素。

content 属性的使用，更多地是与 CSS 选择器结合使用。

2. 示例介绍

【示例 4-19】 给元素添加内容。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>添加 content 内容</title>
<style type="text/css">
```

```
#nav {
    margin:20px auto;
    width:200px;
    border:1px solid #f90;
    padding:20px;
    line-height:22px;
    font-size:18px;
}
#nav a {
    display:block;
    font-size:12px;
    line-height:22px;
    color:#00F;
}
/* 筛选链接地址，添加不同的内容 */
a[href$=html]:before {
    content:"网页：";                /* 指定添加内容 */
}
a[href$=jpg]:before {
    content:"图片：";
}
a[href$=doc]:before {
    content:"Word 文档：";           /* 指定添加内容 */
}
a[href$=pdf]:before {
    content:"PDF 文档：";            /* 指定添加内容 */
}
</style>
</head>
<body>
<div id="nav">
    <a href="Code4-17.html">登录页面</a>
    <a href="images/IL5.jpg">杭州风光</a>
    <a href="images/test.doc">参考资料</a>
    <a href="images/ Pattern.pdf">设计模式培训</a>
</div>
</body>
</html>
```

运行结果如图 4-25 所示。



图 4-25　使用 content 属性给元素添加内容

代码分析：在示例 4-19 中，通过筛选链接地址，为不同的链接添加不同的内容。如图 4-25 所示，每个链接中的冒号及其左边的部分，为添加的内容。

也可以使用 content 属性值 url（<url>）添加图片内容，调整部分样式表内容如下：

```
<style type="text/css">
…
/* 筛选链接地址，添加不同的内容 */
a[href$=html]:after {
    content:url(images/web.png);
}
a[href$=jpg]:after {
    content:url(images/img.png);            /* 指定图片作为添加内容 */
}
a[href$=doc]:after {
    content:url(images/doc.png);            /* 指定图片作为添加内容 */
}
a[href$=pdf]:after {
    content:url(images/pdf.png);                /* 指定图片作为添加内容 */
}
</style>
```

运行结果如图 4-26 所示。



图 4-26　指定图片作为添加内容

也可以使用 content 属性值 attr（<attribute>）把元素属性的属性值作为添加内容，调整部分样式表内容如下：

```
<style type="text/css">
…
/* 添加内容 */
a:after {
    content:attr(href);     /* 元素 a 的 href 属性值作为添加内容 */
}
</style>
```

运行结果如图 4-27 所示。

也可以使用 content 属性值 counter（<counter >），生成项目符号作为添加内容。这其中还需要借助 CSS 的另一属性 counter-increment（这里不详细讲解），来定义计数器。调整样式表如下：



图 4-27　元素的属性值作为添加内容

```
<style type="text/css">
#nav {
    margin:5px auto;
    width:200px;
    border:1px solid #f90;
    padding:20px;
    line-height:22px;
    font-size:18px;
}
#nav a {
    display:block;
    font-size:12px;
    line-height:22px;
    color:#00F;
    counter-increment:mycounter;          /* 定义计数器 */
}
a:before {
    content:counter(mycounter)".";        /* 生成项目符号 */
}
</style>
```

运行结果如图 4-28 所示。



图 4-28　生成项目符号作为添加内容

4.3.5　实验室：设计一个省份选择盘

在网页中通常会有选择省份的下拉列表，列出了所有的省份供用户选择。但因列表较长，选择起来极不友好，比较好的方案就是设计一个省份选择盘。下面我们用前面学过的知识，做一个实验性的示例：设计一个省份选择盘。

1. 案例简介

本节介绍的案例，只有一个选择盘的区域，用户可以调整尺寸；选择盘内部包含 31 个省份的链接，每个链接都有一个编号；当链接获取焦点时，会有一个外轮廓线；当鼠标经过链接时，链接被修饰为按钮风格。页面效果如图 4-29 所示，展示了链接获取焦点的样式和鼠标经过的样式。下面就逐步设计该网页。



图 4-29　省份选择盘界面效果

2. 设计网页元素

页面元素的设计比较简单，一个 div 元素作为选择盘，内部包含 31 个省份的链接。

【示例 4-20】 设计一个省份选择盘。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>省份选择盘</title>
</head>
<body>
<div id="disk"> <a href="#">北京</a> <a href="#">上海</a> <a href="#">天津
</a> <a href="#">重庆</a> <a href="#">安徽</a> <a href="#">福建</a> <a
href="#">甘肃</a> <a href="#">广东</a> <a href="#">广西</a> <a href="#">贵
州</a> <a href="#">海南</a> <a href="#">河北</a> <a href="#">河南</a> <a
href="#">湖北</a> <a href="#">湖南</a> <a href="#">吉林</a> <a href="#">江
苏</a> <a href="#">江西</a> <a href="#">辽宁</a> <a href="#">宁夏</a> <a
href="#">青海</a> <a href="#">山东</a> <a href="#">山西</a> <a href="#">陕
西</a> <a href="#">四川</a> <a href="#">西藏</a> <a href="#">新疆</a> <a
```

```
href="#">云南</a> <a href="#">浙江</a><a href="#">黑龙江</a> <a href="#">内
蒙古</a> </div>
</body>
</html>
```

3. 样式表设计

设计选择盘允许用户改变尺寸。内部链接元素浮动显示，可以随选择盘的尺寸改变而自适应布局。内部链接通过样式表添加项目符号。

```
<style type="text/css">
#disk {
    width:340px;
    padding:10px;
    resize:both;                      /* 允许改变尺寸 */
    overflow:auto;                    /* 溢出处理 */
    border:1px solid #f90;
    line-height:22px;
}
#disk a {
    display:block;                    /* 显示为块结构 */
    float:left;                       /* 左浮动 */
    width:60px;
    text-align:center;
    text-decoration:none;
    font-size:12px;
    line-height:20px;
    margin:3px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    counter-increment:mycounter;      /* 定义计数器 */
}
#disk a:focus {
    outline:2px solid #fc6;           /* 链接焦点设置外轮廓 */
    outline-offset:2px;
}
#disk a:hover {
    -webkit-appearance:button;        /* 鼠标经过链接，修饰为按钮风格 */
    -moz-appearance:button;           /* 鼠标经过链接，修饰为按钮风格 */
    appearance:button;                /* 鼠标经过链接，修饰为按钮风格 */
}
#disk a:before {
    content:counter(mycounter)"."; /* 生成并添加项目符号 */
}
</style>
```

至此，完成了省份选择盘的页面设计，运行结果如图 4-29 所示。

4.4　小　　结

本章讲解了 CSS 3 新增的盒布局和界面设计方面的一些属性。重点讲解了盒布局的思想及其相关属性，盒子阴影是比较常用的属性，也是本章的重点。其中盒布局的思想比较

难以理解，与传统的布局方法有很大出入。给元素添加内容有极大的灵活性，也不宜理解，需要读者多体会。

下一章将介绍 CSS 3 新增的另外一种布局——多列布局。

4.5　习　　题

【习题 1】如何开启盒布局？

【习题 2】请选择可用于盒布局的属性（多选）：

A. box-orient　　B. box-direction　　C. box-ordinal-group

D. box-shadow　　E. box-sizing　　F. box-flex

【习题 3】请选择 box-shadow 属性的正确设置（多选）：

A. box-shadow:5px 5px 5px #333;　　B. box-shadow:-5px -5px 5px #00f;

C. box-shadow:0 0 0 5px #333;　　D. box-shadow: inset 5px 5px 5px #333;

【习题 4】下面的选项中，哪些属于 box-sizing 属性的值（多选）：

A. content-box　　B. box　　C. border-box

D. inherit　　E. padding-box　　F. margin-box

【习题 5】下面的选项中，哪个属性是用于定义外轮廓线的（多选）：

A. resize　　B. outline　　C. appearance

D. content　　E. outline-offset　　F. outline-style

第 5 章　你一直期待的多列布局

在我们经常阅读的报纸或杂志中，通常一个版面会分成多个列进行排版。在传统的网页设计中，会使用表格布局或浮动布局等方式，但总会遇到同样的错位问题，也因此需要反复地调整，但仍然不够完美。针对这种情况，你是否在期待一种更好的解决方法？CSS 3 提供了新的多列布局，可以直接定义列数、列宽等，也可以定义列与列之间的间距、间隔线等，还可以定义栏目跨列和栏目高度等。本章将详细讲解多列布局的基本属性及其使用方法。

5.1　多列布局基础

CSS 3 新增的多列布局，可以从多个方面去设置：多列的列数、每列的宽度、列与列之间的距离、列与列之间的间隔线样式、跨多列设置和列的高度设置等，下面逐步讲解。

5.1.1　多列属性 columns

CSS 3 新增的 columns 属性，用于快速定义多列的列数目和每列的宽度。基于 webkit 内核的替代私有属性是-webkit-columns，gecko 内核的浏览器暂不支持。

1. 参数说明

columns 属性的语法如下：

```
columns:<column-width> || <column-count> ;
```

取值说明如下。

- <column-width>：定义每列的宽度。
- <column-count>：定义多列的列数。

在实际布局的时候，所定义的多列的列数是最大列数。当外围宽度不足时，多列的列数会适当减少，而每列的宽度会自适应宽度，填满整个范围区域。

2. 示例介绍

【示例 5-1】 把一篇文章分成多列显示。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
```

```
<title>多列属性 columns</title>
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-columns:200px 3;    /* 设计为 3 列，每列宽度为 200px */
    columns:200px 3;            /* 设计为 3 列，每列宽度为 200px */
}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em; font-size:12px;line-height:20px;}
</style>
</head>
<body>
<h1>背影</h1>
<h2>朱自清</h2>
<p>我与父亲不相见已二年余了，我最不能忘记的是他的背影。</p>
<p>那年冬天，祖母死了，父亲的差使也交卸了，正是祸不单行的日子。我从北京到徐州打算跟着父亲奔丧回家。到徐州见着父亲，看见满院狼籍的东西，又想起祖母，不禁簌簌地流下眼泪。父亲说："事已如此，不必难过，好在天无绝人之路！"</p>
<p>回家变卖典质，父亲还了亏空；又借钱办了丧事。这些日子，家中光景很是惨淡，一半因为丧事，一半因为父亲赋闲。丧事完毕，父亲要到南京谋事，我也要回北京念书，我们便同行。</p>
<p>到南京时，有朋友约去游逛，勾留了一日；第二日上午便须渡江到浦口，下午上车北去。父亲因为事忙，本已说定不送我，叫旅馆里一个熟识的茶房陪我同去。他再三嘱咐茶房，甚是仔细。但他终于不放心，怕茶房不妥帖；颇踌躇了一会。其实我那年已二十岁，北京已来往过两三次，是没有什么要紧的了。他踌躇了一会，终于决定还是自己送我去。我两三回劝他不必去；他只说："不要紧，他们去不好！"</p>
<p>我们过了江，进了车站。我买票，他忙着照看行李。行李太多了，得向脚夫行些小费才可过去。他便又忙着和他们讲价钱。我那时真是聪明过分，总觉他说话不大漂亮，非自己插嘴不可，但他终于讲定了价钱；就送我上车。他给我拣定了靠车门的一张椅子；我将他给我做的紫毛大衣铺好座位。他嘱我路上小心，夜里要警醒些，不要受凉。又嘱托茶房好好照应我。我心里暗笑他的迂；他们只认得钱，托他们只是白托！而且我这样大年纪的人，难道还不能料理自己么？唉，我现在想想，那时真是太聪明了！</p>
</body>
</html>
```

运行结果如图 5-1 所示。

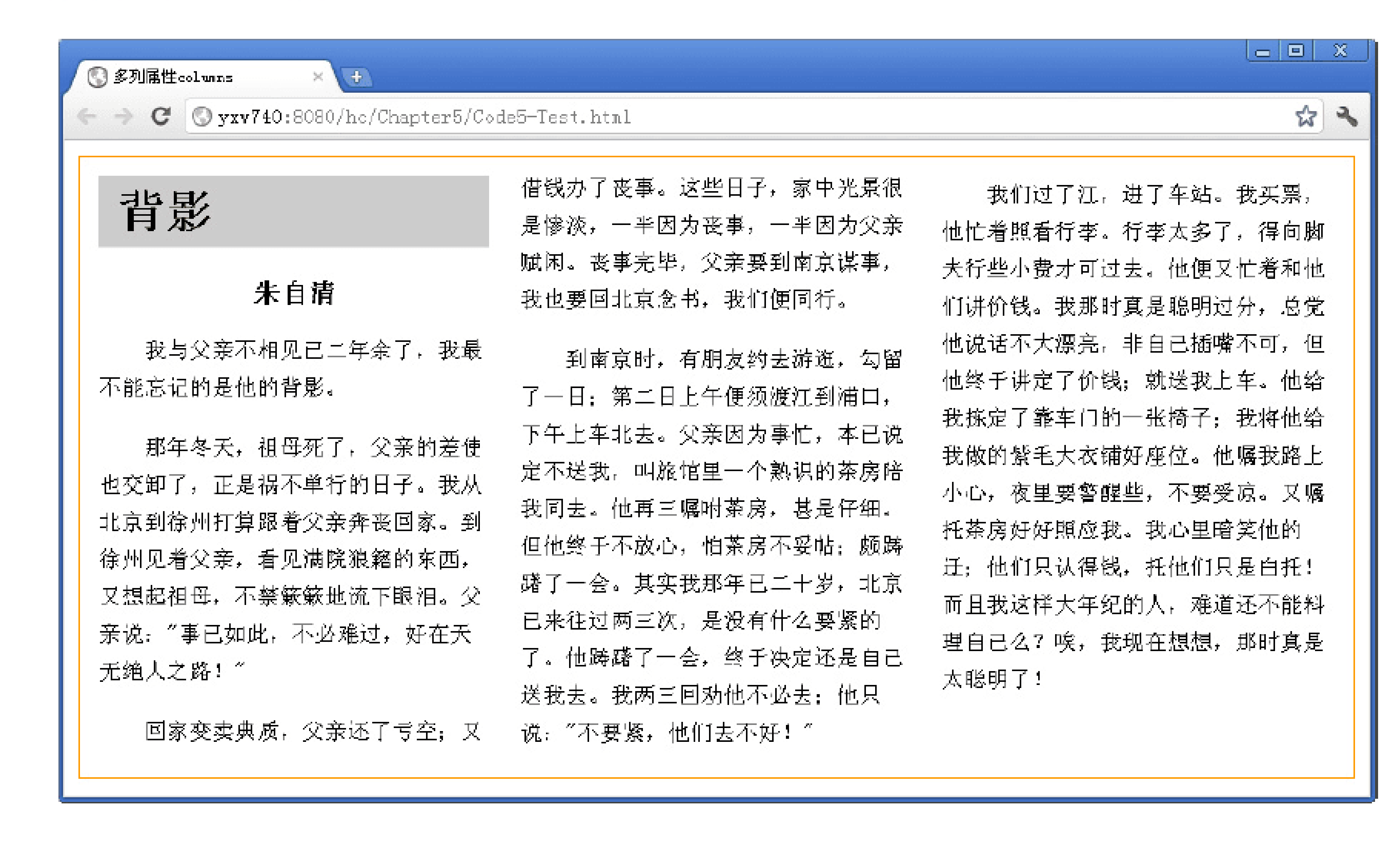


图 5-1　一篇文章分成多列的显示效果

代码分析：在示例 5-1 中，仅仅设置了 columns 属性，即实现了分列布局，且每列的高度尽可能一致，如图 5-1 所示。

如果缩小浏览器窗体的宽度，则文章会变成 2 列或者 1 列，每列的高度尽可能一致，而每列的宽度会自适应分配，不一定是 200px。

5.1.2　列宽属性 column-width

CSS 3 新增的 column-width 属性，用于定义多列布局中每列的宽度。基于 webkit 内核的替代私有属性是-webkit-column-width，基于 gecko 内核的替代私有属性是-moz-column-width。

1. 参数说明

column-width 属性的语法如下：

```
column-width : auto | <length> ;
```

取值说明如下。

- auto：列的宽度由浏览器决定。
- <length>：直接指定列的宽度，该值是由浮点数和单位标识符组成的长度值，不可以为负值。

2. 示例介绍

在示例 5-1 的基础上，调整 body 标签的样式表，设置 column-width 属性。

【示例 5-2】 仅设置列的宽度。

```
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-width:200px;     /* 每列宽度为 200px */
-moz-column-width:200px;            /* 每列宽度为 200px */
    column-width:200px;             /* 每列宽度为 200px */
}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em; font-size:12px;line-height:20px;}
</style>
```

运行结果如图 5-1、图 5-2 所示。

代码分析：在示例 5-2 中，仅设置了每列的宽度。当窗口的大小改变时，列数会及时调整，列数不固定。

5.1.3　列数属性 column-count

CSS 3 新增的 column-count 属性用于定义多列布局中的列数目。基于 webkit 内核的替代私有属性是-webkit-column-count，基于 gecko 内核的替代私有属性是-moz-column-count。



图 5-2　缩小窗口宽度变成两列

1. 参数说明

column-count 属性的语法如下：

```
column-count : auto | <number> ;
```

取值说明如下。

- ❑ auto：列的数目由其他属性决定，如 column-width。
- ❑ <number>：直接指定列的数目，取值为大于 0 的整数，决定了多列的最大列数。

2. 示例介绍

在示例 5-1 的基础上，调整 body 标签的样式表，设置 column-count 属性。

【示例 5-3】 仅设置列的数目。

```
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-count:3;     /* 指定列的数目 */
    -moz-column-count:3;        /* 指定列的数目 */
    column-count:3;             /* 指定列的数目 */
}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em;font-size:12px;line-height:20px;}
</style>
```

运行结果如图 5-1、图 5-3 所示。

代码分析：在示例 5-3 中，仅设置了多列的列数目。当窗口的大小改变时，列宽会及时调整，列数固定不变。

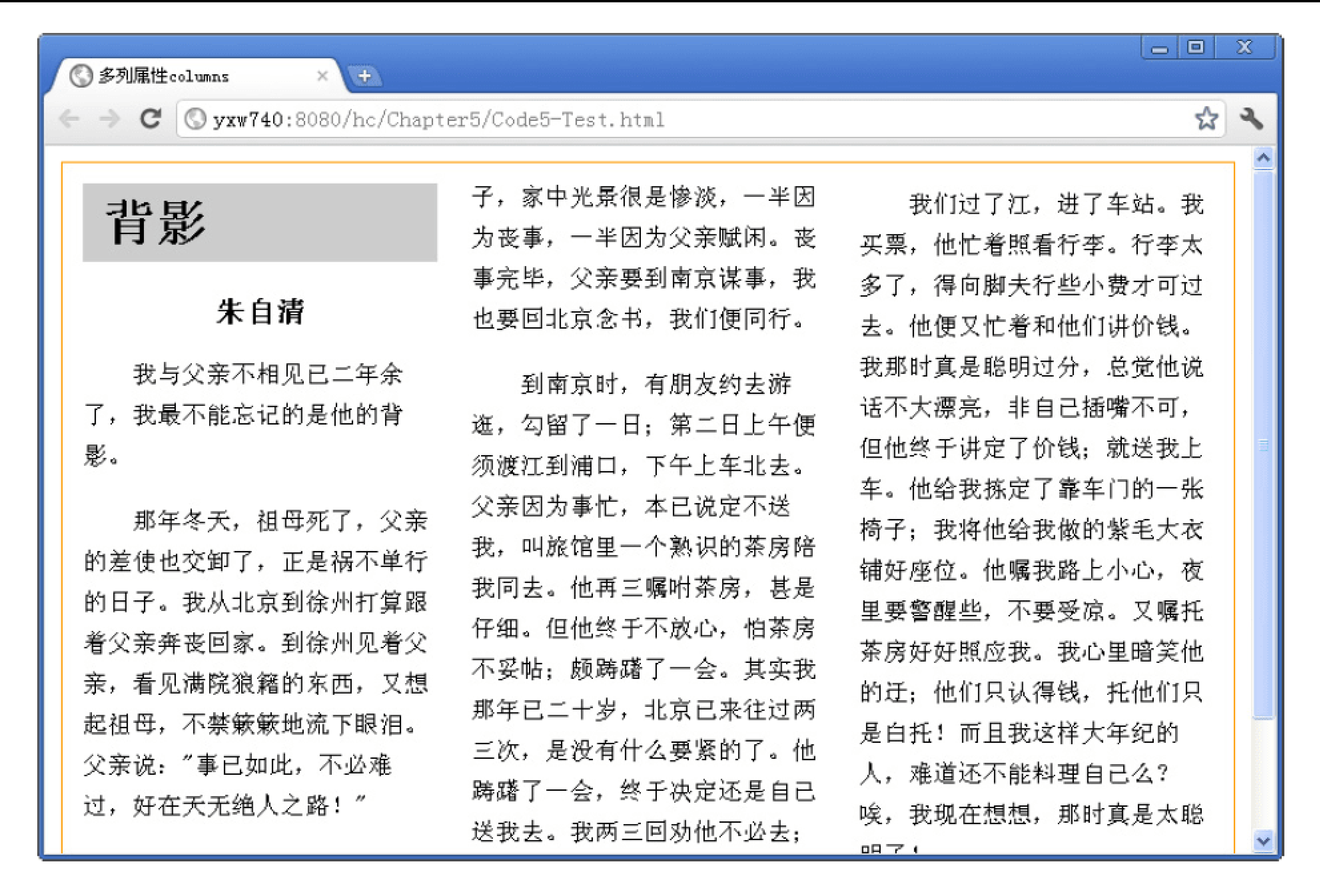


图 5-3　缩小窗口仍然保持 3 列

5.1.4　列间距属性 column-gap

CSS 3 新增的 column-gap 属性，用于定义多列布局中列与列之间的距离。基于 webkit 内核的替代私有属性是-webkit-column-gap，基于 gecko 内核的替代私有属性是-moz-column-gap。

1. 参数说明

column-gap 属性的语法如下：

```
column-gap : normal | <length> ;
```

取值说明如下。

- ❑ normal：默认值，由浏览器默认的列间距，一般是 1em。
- ❑ <length>：指定列与列之间的距离，由浮点数字和单位标识符组成，不可为负值。

2. 示例介绍

在多列布局中，设置 body 标签的 column-gap 属性，调整为较宽松的列间距。

【示例 5-4】 较宽松的列间距。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>宽松的列间距</title>
<style type="text/css">
body {
    border:1px solid #f90;
```

```
    padding:10px;
    -webkit-column-count:3;          /* 指定列的数目 */
    -moz-column-count:3;             /* 指定列的数目 */
    column-count:3;                  /* 指定列的数目 */
    -webkit-column-gap:3em;          /* 指定列间距为 3em */
    -moz-column-gap:3em;             /* 指定列间距为 3em */
    column-gap:3em;                  /* 指定列间距为 3em */
}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em; font-size:12px;line-height:20px;}
</style>
<body>
<h1>背影</h1>
<h2>朱自清</h2>
<p>…</p>
</body>
</html>
```

运行结果如图 5-4 所示。

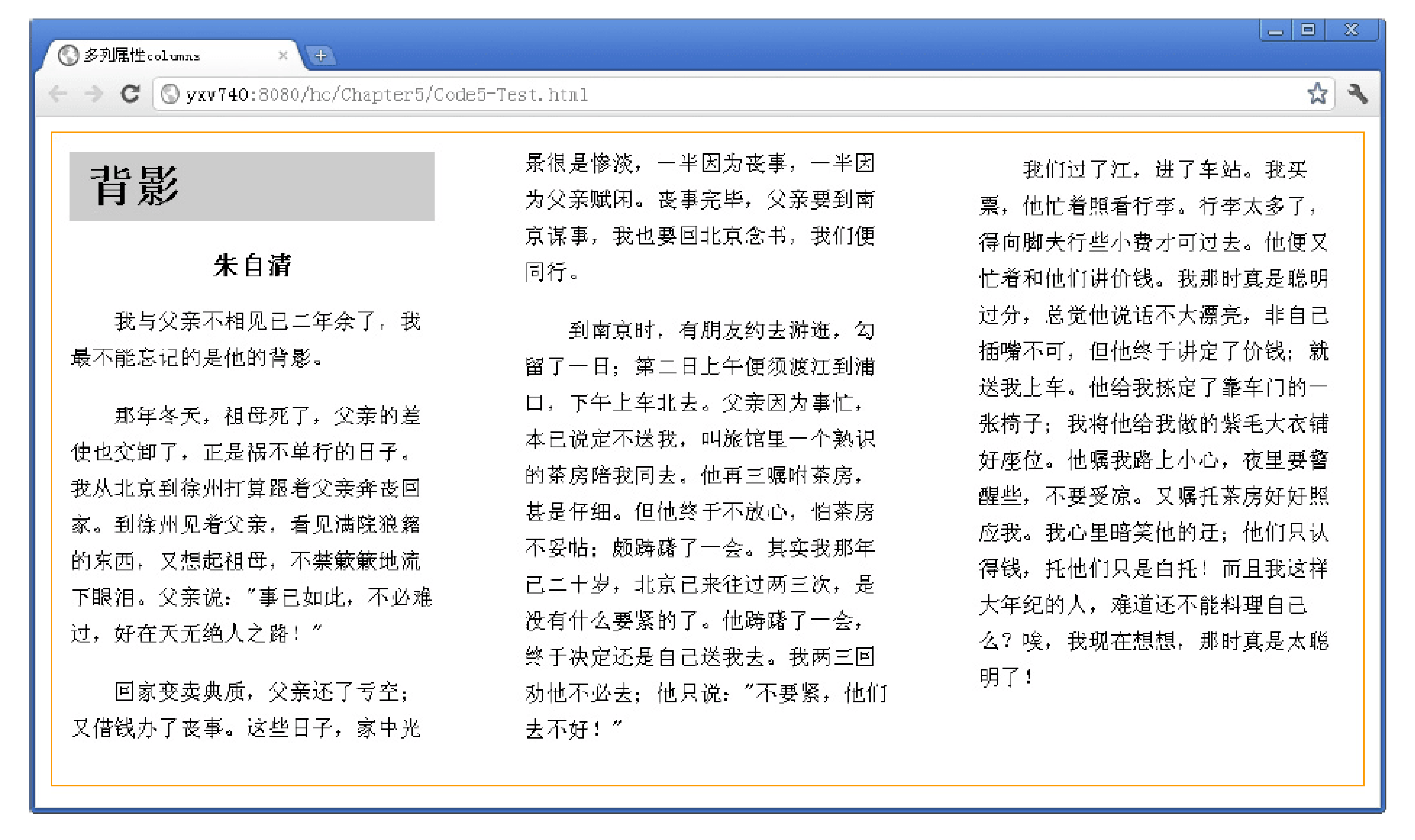


图 5-4　较为宽松的列间距

代码分析：在示例 5-4 中，在 3 列的基础上，设置了 column-gap 值为 3em，列与列之间的距离增加了很多，变得较为宽松了。

5.1.5　定义列分隔线——column-rule 属性

CSS 3 新增的 column-rule 属性，在多列布局中，用于定义列与列之间的分隔线。基于 webkit 内核的替代私有属性是-webkit-column-rule，基于 gecko 内核的替代私有属性是-moz-column-rule。

1. 参数说明

column-rule 属性的语法如下：

```
column-rule : [column-rule-width] || [column-rule-style] || [column-rule-color] ;
```

取值说明如下。

- <column-rule-width>：定义分隔线的宽度。
- <column-rule-style>：定义分隔线的样式风格。
- <column-rule-color>：定义分隔线的颜色。

2. 派生的子属性

下面介绍关于派生的子属性。

- column-rule 属性是一个复合的属性，包含 3 个子属性，分别定义分隔线的宽度、样式风格和颜色。
- column-rule-width 子属性：定义分隔线宽度，为任意包含单位的长度，不可为负值。
- column-rule-style 子属性：定义分隔线样式风格，取值范围与 border-style 相同，包括 none、dotted、dashed、solid、double、groove、ridge、inset、outset、inherit。
- column-rule-color 子属性：定义分隔线的颜色，为任意用于 CSS 的颜色值，也包括半透明颜色。

column-rule 属性及其子属性的使用方法，与 border 属性及其子属性相同。对于 webkit 内核的浏览器，column-rule 属性及其子属性需要增加前缀“-webkit-”；对于 gecko 内核的浏览器，column-rule 属性及其子属性需要增加前缀“-moz-”。

3. 示例介绍

在多列布局中，设置列与列之间的分隔线。

【示例 5-5】 列与列之间的分隔线。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>分隔线</title>
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-count:3;                /* 指定列的数目 */
    -moz-column-count:3;                   /* 指定列的数目 */
    column-count:3;                        /* 指定列的数目 */
    -webkit-column-rule:1px solid #666;    /* 指定列间距为 3em */
    -moz-column-rule:1px solid #666;       /* 指定列间距为 3em */
    column-rule:1px solid #666;            /* 指定列间距为 3em */
}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em; font-size:12px;line-height:20px;}
</style>
<body>
<h1>背影</h1>
<h2>朱自清</h2>
<p>…</p>
```

```
</body>
</html>
```

运行结果如图 5-5 所示。



图 5-5　列与列的分隔线

代码分析：在示例 5-5 中，仍然定义了 3 列。我们直接通过复合属性 column-rule，设置了分隔线的宽度、样式和颜色。显示效果如图 5-5 所示。

分隔线的样式如同边框样式一样可以调整，当然也可以使用子属性进行设置。调整样式表如下：

```
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-count:3;                    /* 指定列的数目 */
    -moz-column-count:3;                       /* 指定列的数目 */
    column-count:3;                            /* 指定列的数目 */
    -webkit-column-rule-width:1px;             /* 设置分隔线的宽度 */
    -moz-column-rule-width:1px;                /* 设置分隔线的宽度 */
    column-rule-width:1px;                     /* 设置分隔线的宽度 */
    -webkit-column-rule-style:dashed;          /* 设置分隔线的样式 */
    -moz-column-rule-style:dashed;             /* 设置分隔线的样式 */
    column-rule-style:dashed;                  /* 设置分隔线的样式 */
    -webkit-column-rule-color:#f00;            /* 设置分隔线的颜色 */
    -moz-column-rule-color:#f00;               /* 设置分隔线的颜色 */
    column-rule-color:#f00;                    /* 设置分隔线的颜色 */}
h1 {font-size:24px;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em; font-size:12px;line-height:20px;}
</style>
```

运行结果如图 5-6 所示。

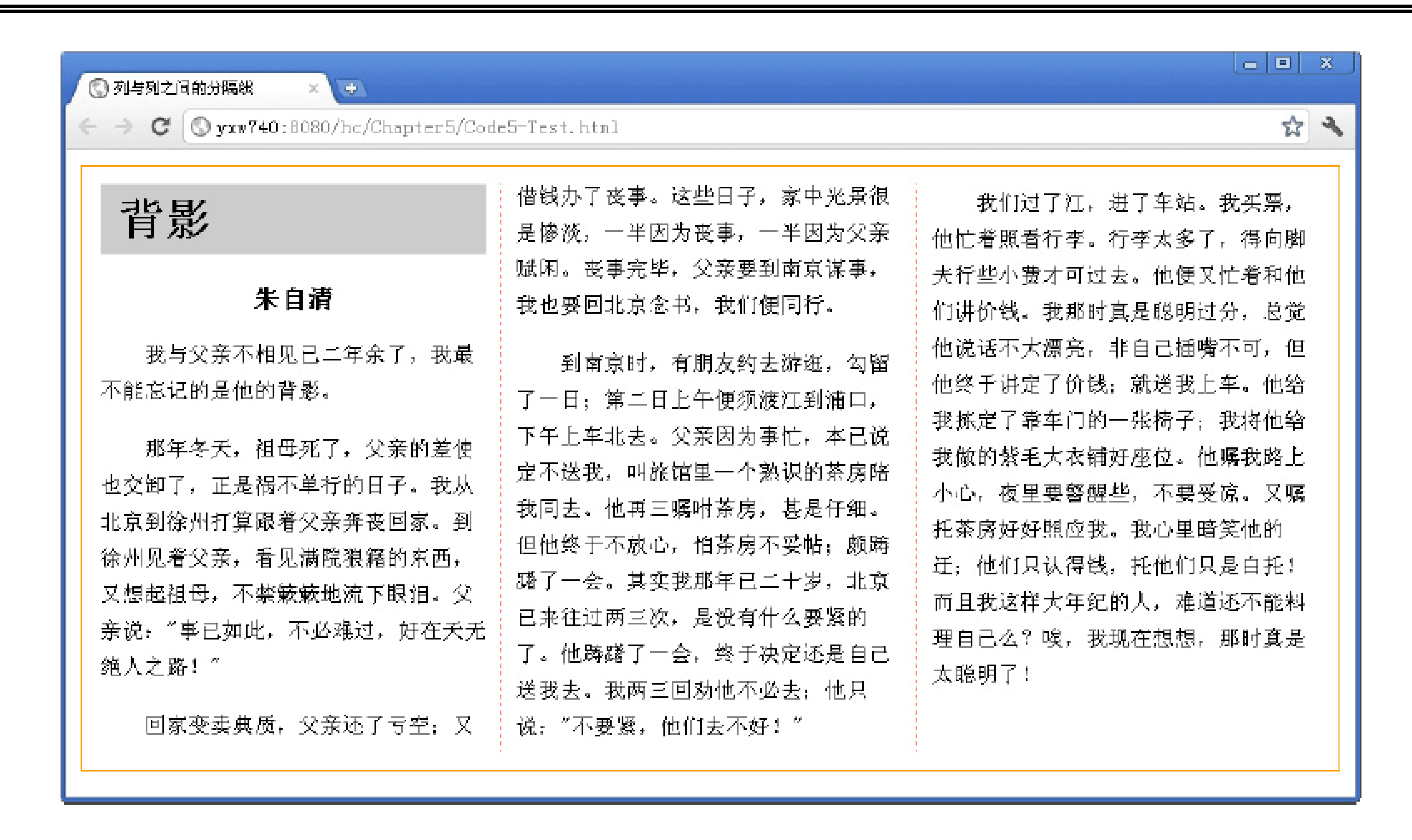


图 5-6　列与列的虚线分隔线

5.1.6　定义横跨所有列——column-span 属性

CSS 3 新增的 column-span 属性，在多列布局中，用于定义元素跨列显示。基于 webkit 内核的替代私有属性是-webkit-column-span，gecko 内核的浏览器暂不支持该属性。

1．参数说明

column-span 属性的语法如下：

```
column-span :1 | all ;
```

取值说明如下。

- ❑ 1：默认值，元素在一列中显示。
- ❑ all：元素横跨所有列显示。

2．示例介绍

在多列布局中，设置标题内容跨列显示。

【示例 5-6】 标题跨列显示。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>标题跨列显示</title>
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-count:3;                   /* 指定列的数目 */
    -moz-column-count:3;                      /* 指定列的数目 */
    column-count:3;                           /* 指定列的数目 */
    -webkit-column-rule:1px solid #666;       /* 设置分隔线 */
    -moz-column-rule:1px solid #666;          /* 设置分隔线 */
```

```
    column-rule:1px solid #666;              /* 设置分隔线 */
}
h1,h2{
    -webkit-column-span:all;                 /* 设置横跨所有列显示 */
    column-span:all;                         /* 设置横跨所有列显示 */
}
h1 {font-size:24px;margin:0;padding:5px 10px;background-color:#CCC;}
h2 {font-size:14px;text-align:center;}
p {text-indent:2em;font-size:12px;line-height:20px;} </style>
</head>
<body>
<h1>背影</h1>
<h2>朱自清</h2>
<p>…</p>
</body>
</html>
```

运行结果如图 5-7 所示。

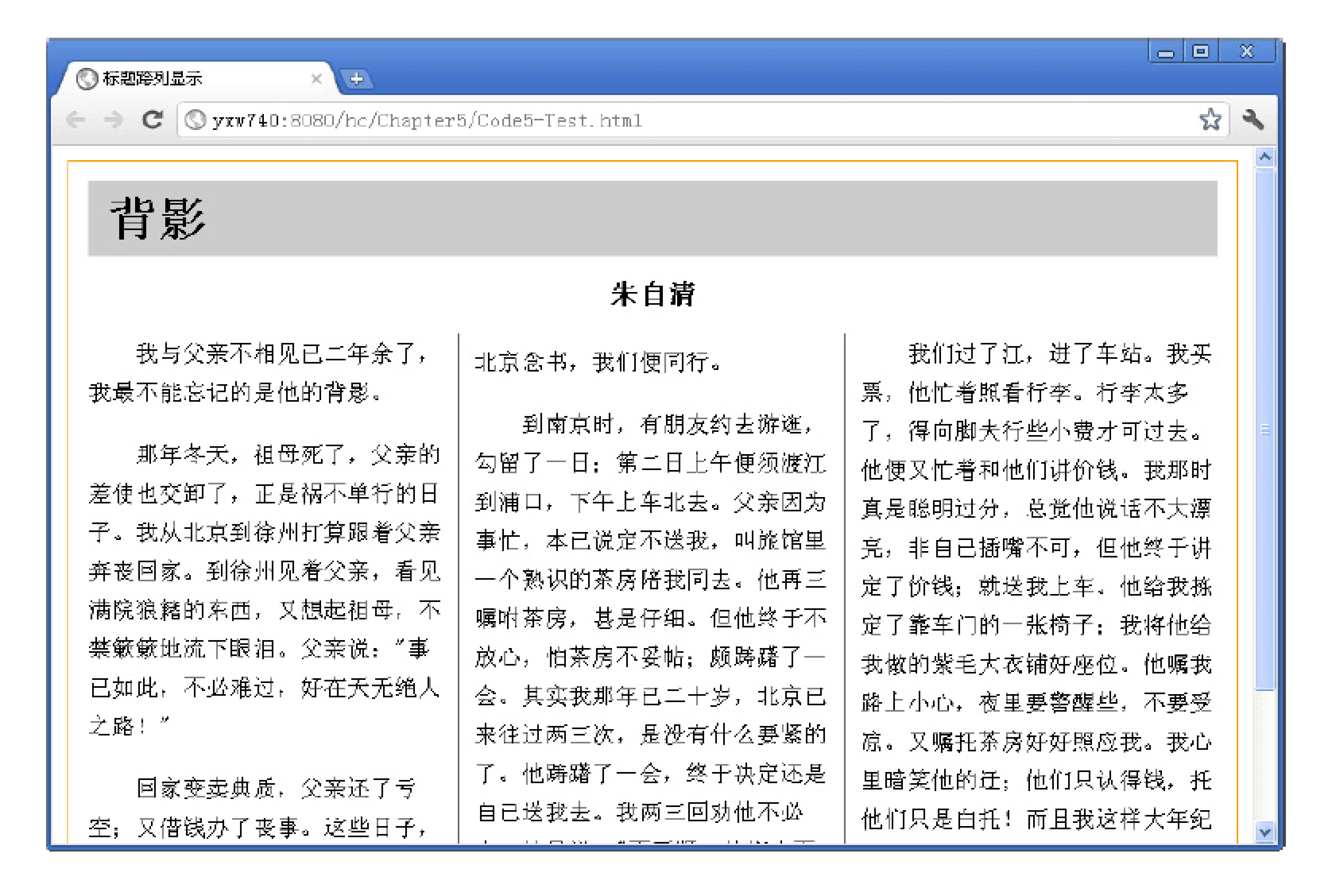


图 5-7　跨列显示的标题栏

代码分析：在示例 5-6 中，统一为标签 h1 和 h2 设置了 column-span 属性值为 all，使其跨所有列显示，如图 5-7 所示。

5.2　实验室：模仿杂志的多列版式

在制作电子杂志的时候，会需要界面布局看起来像杂志，以便从形象上与常规的网页有所区别。传统的技术难点就是多列版式的布局。下面我们就用多列布局来实现电子杂志的多列版式。

1. 案例简介

本节介绍的案例，是一个简易的杂志页面。页面版式分 3 列布局，其中标题横跨所有列显示；文章的开篇内容的第一个字设置为突出显示，内容图文并茂，实现仿杂志效果。

案例效果图如图 5-8 所示。



图 5-8　模仿杂志的多列版式

2. 设计网页元素

网页内容分三类：标题、作者和文章内容。

【示例 5-7】 模仿杂志的多列版式。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>模仿杂志的多列版式</title>
</head>
<body>
<h1>CSS 3 的发展轨迹</h1>
<h2>作者：佚名</h2>
<p class="first">样式表自从 CSS1 的版本之后…</p>
<p>…</p>
</body>
</html>
```

省去了冗长的文字内容。

3. 样式表设计

在 body 标签中设置页面内容使用多列布局，分三列显示。标题横跨所有栏显示，突出文章的第一个字。

```
<style type="text/css">
body {
    border:1px solid #f90;
    padding:10px;
    -webkit-column-count:3;        /* 指定列的数目 */
    -moz-column-count:3;           /* 指定列的数目 */
    column-count:3;                /* 指定列的数目 */
}
h1 {
```

```
    -webkit-column-span:all;        /* 设置横跨所有列显示 */
    -moz-column-span:all;           /* 设置横跨所有列显示 */
    column-span:all;                /* 设置横跨所有列显示 */
    font-size:24px;
    margin:0;
    padding:5px 10px;
    background-color:#e4e4e4;
    text-align:center;
}
h2 {
    font-size:14px;
    text-align:center;
}
p {
    text-indent:2em;
    font-size:12px;
    line-height:20px;
}
.first:first-letter {
    font-size:24px;                 /* 突出第一个字 */
    font-weight:bold;
}
.b {
    font-weight:bold;
}
</style>
```

至此，完成了仿杂志的多列版式设计，运行结果如图 5-8 所示。

5.3　小　　结

本章主要讲解了 CSS 3 多列布局的应用。重点讲解了多列布局中列宽和列数的定义，以及列间距里和列间分隔线的使用方法。本章的难点在于多列布局的思路与传统的布局有很大不同，需要一个思维上的转变，用心领会，其实不难。下一章将介绍 CSS 3 支持的革命性的功能——动画和渐变。

5.4　习　　题

【习题 1】简要列举一下 CSS 3 中有哪几种布局方式。

【习题 2】在多列布局中，如果定义列与列之间的分隔线，应该使用哪个属性（单选）：

A. column-width　　B. column-count　　C. column-gap

D. column-rule　　E. column-fill　　F. column-span

【习题 3】编写一个页面，把一篇文章的内容分成三列显示，并在列与列之间设置分隔线。

第 6 章　酷炫的动画和渐变

在网页设计中，适当地使用动画或者渐变，可以把网页设计得更加生动和友好。在传统的设计中，会借助 Flash 或者 JavaScript 来实现动画，借助图片实现渐变，而 CSS 仅仅是静态地表现元素的效果。不过，CSS 3 将改变我们的思维方式，因为动画和渐变也可以直接用 CSS 来实现了。这些革命性的改变，使得 CSS 具有更强大的可能性。本章就详细地讲解 CSS 3 的 2D 动画和渐变。

6.1　CSS 3 变形基础

变形是实现动画的前提。本节将详细讲解 CSS 3 的 2D 变形基础（3D 变形还未获得广泛的支持）。

6.1.1　元素的变形——transform 属性

CSS 3 新增的 transform 属性，可用于元素的变形，实现元素的旋转、缩放、移动、倾斜等变形效果。基于 webkit 内核的替代私有属性是-webkit-transform；基于 gecko 内核的替代私有属性是-moz-transform；基于 presto 内核的替代私有属性是-o-transform；IE 浏览器的替代私有属性是-ms-transform。transform 属性的语法如下：

```
transform :none | <transform-functions> ;
```

取值说明如下。

- none：默认值，不设置元素变形。
- <transform-functions>：设置一个或多个变形函数。变形函数包括旋转 rotate()、缩放 scale()、移动 translate()、倾斜 skew()、矩阵变形 matrix()等。设置多个变形函数时，用空格间隔。

元素在变形的过程中，仅元素的显示效果变形，实际尺寸并不会因为变形而改变。所以元素变形后，可能会超出原有的限定边界，但不会影响自身尺寸及其他元素的布局。

6.1.2　旋转

rotate()函数用于定义元素在二维空间的旋转。函数的使用语法如下：

```
rotate(<angle>)
```

参数说明：<angle>表示旋转的角度，为带有角度单位标识符的数值，角度单位是 deg。

值为正时，表示顺时针旋转；值为负时，表示逆时针旋转。

【示例 6-1】 旋转的菜单。

下面演示一个旋转菜单的效果。鼠标经过时，菜单会旋转一定角度。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>旋转元素</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
li:hover {
    background-color:#999;                    /* 深灰色 */
}
a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：旋转 */
    -webkit-transform:rotate(30deg);          /* 兼容 webkit 内核 */
    -moz-transform:rotate(30deg);             /* 兼容 gecko 内核 */
    -o-transform:rotate(30deg);               /* 兼容 presto 内核 */
    -ms-transform:rotate(30deg);              /* 兼容 IE9 */
    transform:rotate(30deg);                  /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-1 所示。



图 6-1　鼠标经过时，菜单旋转

代码分析：在示例 6-1 中，设置了 4 个菜单项，为了对比变形的差别，均保留了变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为旋转变形函数 rotate()，旋转角度为 30deg，示例运行结果如图 6-1 所示，鼠标经过菜单时，菜单顺时针旋转 30deg。

提示：针对 transform 属性，不同的浏览器内核都有各自的私有替代属性。IE 的早期版本均可以借助滤镜实现元素的变形。

6.1.3　缩放和翻转

scale()函数用于定义元素在二维空间的缩放和翻转。函数的使用语法如下：

```
scale( <x> , <y> )
```

参数说明如下。

- ❑ <x>：表示元素在水平方向上的缩放倍数。
- ❑ <y>：表示元素在垂直方向上的缩放倍数。
- ❑ <x>、<y>的值可以为整数、负数、小数。当取值的绝对值大于 1 时，表示放大；绝对值小于 1 时，表示缩小。当取值为负数时，元素会被翻转。如果<y>值省略，则说明垂直方向上的缩放倍数与水平方向上的缩放倍数相同。

【示例 6-2】 放大的菜单。

下面演示一个放大菜单的效果。鼠标经过时，菜单会放大一定的倍数。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>缩放元素</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
```

```
    background-color:#e4e4e4;
    text-align:left;
}
li:hover{
    background-color:#999;          /* 深灰色 */
}
a{
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover{
    background-color:#f90;
    color:#FFF;
    /* 变形方式：缩放 */
    -webkit-transform:scale(1.5);  /* 兼容 webkit 内核 */
    -moz-transform:scale(1.5);     /* 兼容 gecko 内核 */
    -o-transform:scale(1.5);       /* 兼容 presto 内核 */
    -ms-transform:scale(1.5);      /* 兼容 IE9 */
    transform:scale(1.5);          /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-2 所示。

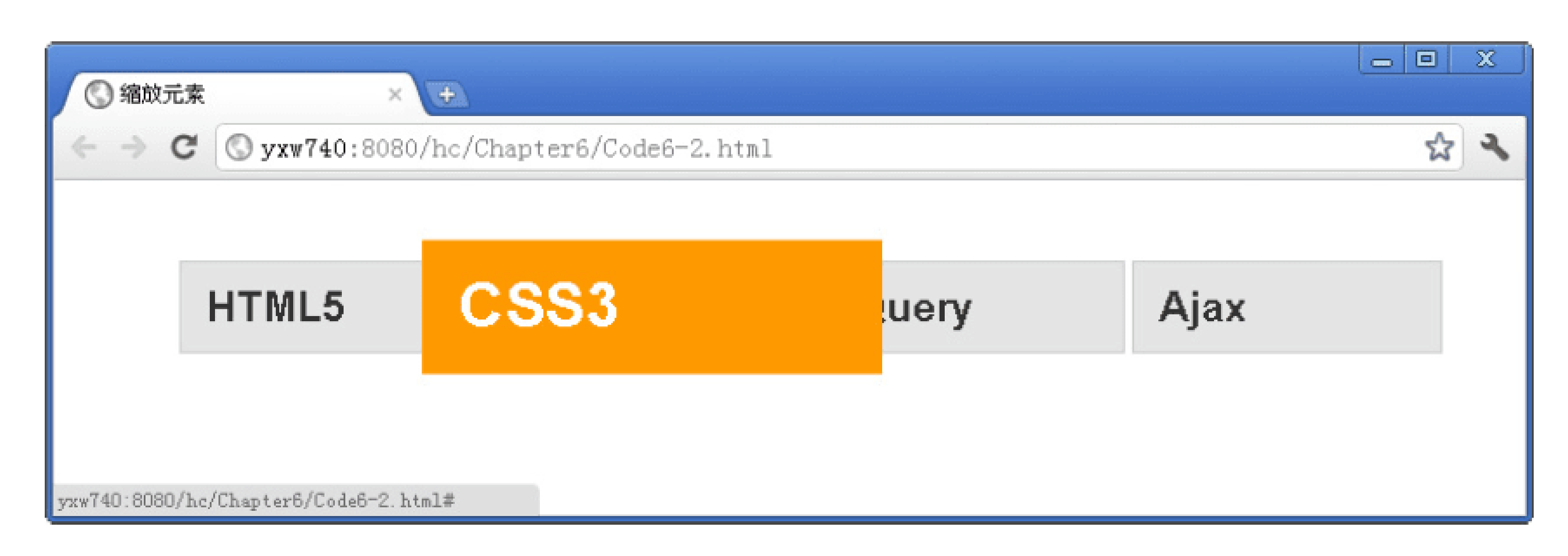


图 6-2　鼠标经过时，菜单放大

代码分析：在示例 6-2 中，在鼠标经过的样式中，设置 transform 属性值为缩放变形函数 scale()，缩放值为 1.5。示例运行结果如图 6-2 所示，鼠标经过菜单时，菜单会放大至 1.5 倍。

如前面 scale()函数语法所述，缩放倍数为负数，元素会翻转；缩放倍数绝对值小于 1，元素会缩小。调整示例 6-2 中的变形样式如下：

```
<style type="text/css">
    … /* 这里省略的样式不变 */
```

```
a:hover{
    background-color:#f90;
    color:#FFF;
    /* 变形方式：缩放 */
    -webkit-transform:scale(0.8,-1.5); /* 兼容 webkit 内核 */
    -moz-transform:scale(0.8,-1.5);     /* 兼容 gecko 内核 */
    -o-transform:scale(0.8,-1.5);       /* 兼容 presto 内核 */
    -ms-transform:scale(0.8,-1.5);      /* 兼容 IE9 */
    transform:scale(0.8,-1.5);          /* 标准写法 */
}
</style>
```

运行结果如图 6-3 所示。



图 6-3　鼠标经过时，菜单缩放

代码分析：在调整后的样式表中，为缩放函数 scale()分别设置了水平方向上的缩放倍数和垂直方向上的缩放倍数。其中水平缩放倍数为小于 1 的正数，表现为元素会在水平向上缩小；垂直缩放倍数为负值且绝对值大于 1，表现为元素会在垂直方向上，放大并且翻转，效果如图 6-3 所示。

6.1.4　移动

translate()函数用于定义元素在二维空间的偏移。函数的使用语法如下：

```
translate (<dx>,<dy>)
```

参数说明如下。

- ❑ <dx>：表示元素在水平方向上的偏移距离。
- ❑ <dy>：表示元素在垂直方向上的偏移距离。
- ❑ <dx>、<dy>的值为带有长度单位标识符的数值，可以为负值和带小数的值。若取值大于 0，则表示向右或向下偏移；若取值小于 0，则表示向左或向上偏移。如果<dy>值省略，则说明垂直方向上的偏移距离默认为 0。

【示例 6-3】　移动的菜单。

下面演示一个移动菜单的效果。鼠标经过时，菜单会发生一定的偏移。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>移动元素</title>
```

```
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
li:hover { background-color:#999; }            /* 深灰色 */
a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：移动 */
    -webkit-transform:translate(10px,5px);     /* 兼容 webkit 内核 */
    -moz-transform:translate(10px,5px);        /* 兼容 gecko 内核 */
    -o-transform:translate(10px,5px);          /* 兼容 presto 内核 */
    -ms-transform:translate(10px,5px);         /* 兼容 IE9 */
    transform:translate(10px,5px);             /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-4 所示。



图 6-4　鼠标经过时，菜单移动一定的距离

代码分析：在示例 6-3 中，为了对比变形的差别，均保留菜单项变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为移动变形函数 translate()，水平方向移动 10px，垂直方向移动 5px，示例运行结果如图 6-4 所示，鼠标经过菜单时，菜单会移动一定的距离。

如前面 translate()函数语法所述，若取值小于 0，则表示向左或向上偏移；如果第二个参数值省略，则说明垂直方向上的偏移距离默认为 0。调整示例 6-3 中的变形样式如下：

```
<style type="text/css">
    … /* 这里省略的样式不变 */
a:hover{
    background-color:#f90;
    color:#FFF;
    /* 变形方式：缩放 */
    -webkit-transform: translate(-10px);    /* 兼容 webkit 内核 */
    -moz-transform: translate(-10px);       /* 兼容 gecko 内核 */
    -o-transform: translate(-10px);         /* 兼容 presto 内核 */
    -ms-transform: translate(-10px);        /* 兼容 IE9 */
    transform: translate(-10px);            /* 标准写法 */
}
</style>
```

运行结果如图 6-5 所示。

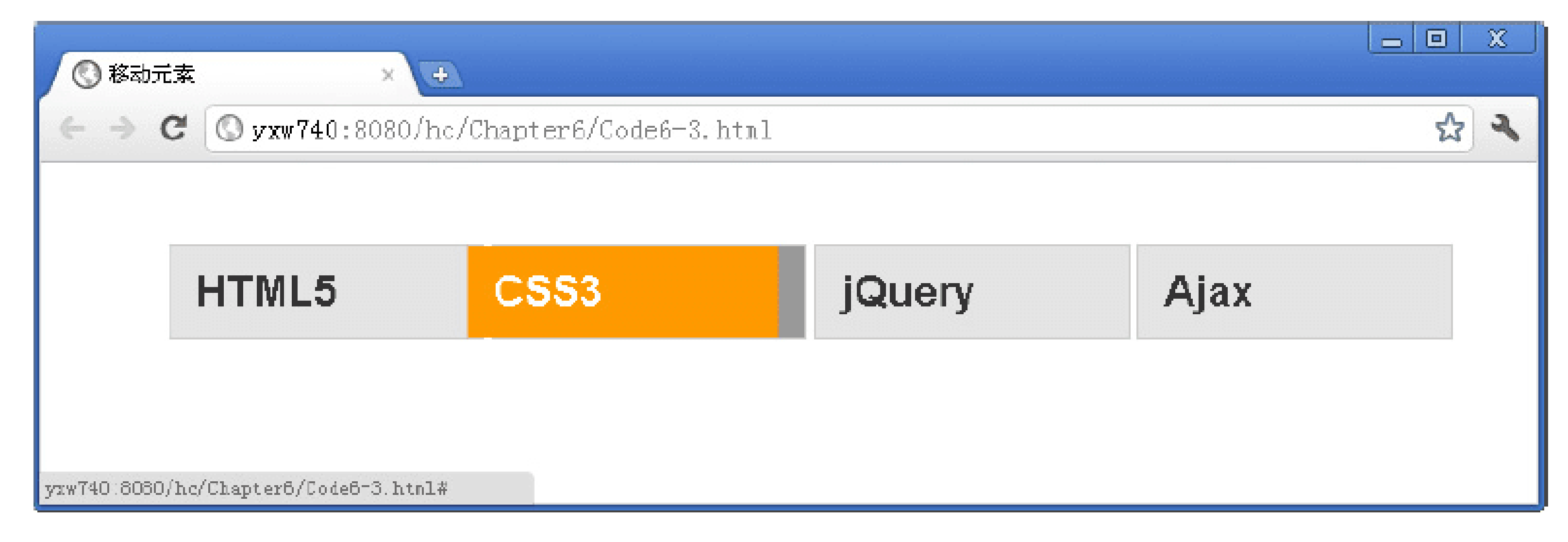


图 6-5　鼠标经过时，菜单缩放

代码分析：在调整后的样式表中，为移动函数 translate()仅设置了一个负值，表现为元素会在水平向上向左偏移，垂直方向上默认为 0，即不偏移。运行效果如图 6-5 所示。

6.1.5　倾斜

skew()函数用于定义元素在二维空间的倾斜变形。函数的使用语法如下：

```
skew (<angleX>,<angleY>)
```

参数说明如下。

- <angleX>：表示元素在空间 x 轴上的倾斜角度。
- <angleY>：表示元素在空间 y 轴上的倾斜角度。
- <angleX>、<angleY>的值为带有角度单位标识符的数值，角度单位是 deg。值为正时，表示顺时针旋转；值为负时，表示逆时针旋转。如果<angleY>值省略，则说明垂直方向上的倾斜角度默认为 0deg。

【示例 6-4】 倾斜的菜单。

下面演示一个倾斜菜单的效果。鼠标经过时，菜单倾斜。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>倾斜元素</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
li:hover { background-color:#999; }        /* 深灰色 */
a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：倾斜 */
    -webkit-transform:skew(-30deg);         /* 兼容 webkit 内核 */
    -moz-transform:skew(-30deg);            /* 兼容 gecko 内核 */
    -o-transform:skew(-30deg);              /* 兼容 presto 内核 */
    -ms-transform:skew(-30deg);             /* 兼容 IE9 */
    transform:skew(-30deg);                 /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-6 所示。

代码分析：在示例 6-4 中，为了对比变形的差别，仍然保留菜单项变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为倾斜变形函数 skew()，水平方向倾斜角度为 30deg，垂直方向倾斜角度默认为 0deg，示例运行结果如图 6-6 所示，鼠标经过菜单时，菜单会倾斜。

根据函数 skew()的语法介绍，也可以设置空间 y 轴上的倾斜角度。调整示例 6-4 中的变形样式如下：



图 6-6　鼠标经过时，菜单倾斜

```
<style type="text/css">
    … /* 这里省略的样式不变 */
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：倾斜 */
    -webkit-transform:skew(30deg,-10deg);  /* 兼容 webkit 内核 */
    -moz-transform:skew(30deg,-10deg);     /* 兼容 gecko 内核 */
    -o-transform:skew(30deg,-10deg);       /* 兼容 presto 内核 */
    -ms-transform:skew(30deg,-10deg);      /* 兼容 IE9 */
    transform:skew(30deg,-10deg);          /* 标准写法 */
}
</style>
```

运行结果如图 6-7 所示。



图 6-7　鼠标经过时，菜单倾斜

代码分析：在调整后的样式表中，为倾斜函数 skew()仅设置了两个角度值，元素会沿着 x 轴和 y 轴表现为不同程度的倾斜。运行效果如图 6-7 所示。

6.1.6　矩阵变形

matrix()函数用于定义元素在二维空间的矩阵变形。函数的使用语法如下：

```
matrix (<m11>,<m12>,<m21>,<m22>,<dx>,<dy>)
```

参数说明：该函数中的 6 个参数均为可计算的数值，组成一个变形矩阵，与当前元素旧的参数组成的矩阵进行乘法运算，形成新的矩阵，元素的参数被改变。该变形矩阵的形式如下：

```
| m11   m21   dx |
| m12   m22   dy |
| 0      0     1  |
```

关于详细的矩阵变形原理，需要掌握矩阵的相关知识，具体可参考数学及图形学相关资料。不过这里可以先通过几个特例了解其大概的使用方法。前面已经讲过的移动、缩放和旋转这些变换相对容易理解，其实都可看作矩阵变形的特例。

- 旋转 rotate(A)，相当于矩阵 matrix(cosA,sinA,-sinA,cosA,0,0)。
- 缩放 scale(sx , sy)，相当于矩阵 matrix(sx,0,0,sy,0,0)。
- 移动 translate (dx, dy) ，相当于矩阵 matrix(1,0,0,1,dx,dy)。

可见，使用矩阵，可以使元素的变形更加灵活。

【示例 6-5】 矩阵变形的菜单。

下面我们使用矩阵演示一个包含多种变形的菜单。鼠标经过时，菜单会发生矩阵变形。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>矩阵变形的元素</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
li:hover { background-color:#999; }                    /* 深灰色 */
}
a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：矩阵变形 */
    -webkit-transform:matrix(0.866,0.5,0.5,-0.866,10,10);
                                                       /* 兼容 webkit 内核 */
    -moz-transform:matrix(0.866,0.5,0.5,-0.866,10,10);
                                                       /* 兼容 gecko 内核 */
    -o-transform:matrix(0.866,0.5,0.5,-0.866,10,10);/* 兼容 presto 内核 */
    -ms-transform:matrix(0.866,0.5,0.5,-0.866,10,10);/* 兼容 IE9 */
    transform:matrix(0.866,0.5,0.5,-0.866,10,10);  /* 标准写法 */
}
</style>
</head>
<body>
<ul>
```

```
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-8 所示。

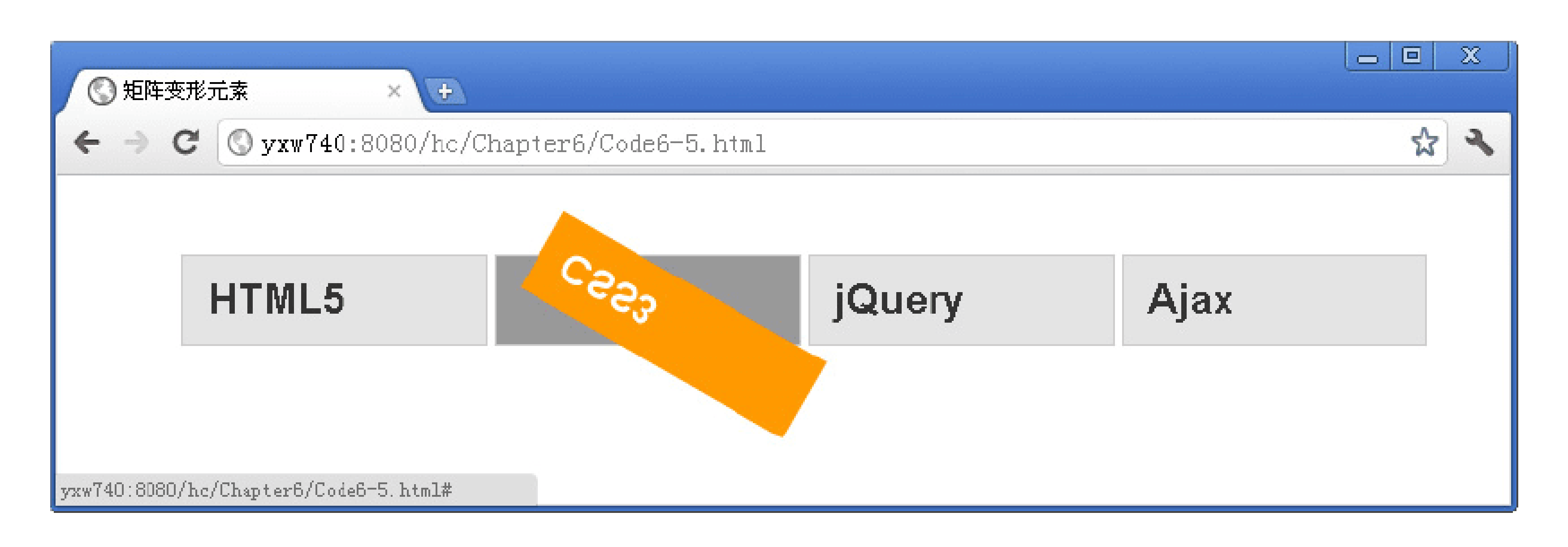


图 6-8　鼠标经过时，菜单发生矩阵变形

代码分析：在示例 6-5 中，为了对比变形的差别，仍然保留菜单项变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为矩阵变形函数 matrix()，示例运行结果如图 6-8 所示，鼠标经过菜单时，菜单会变形，其中变形的效果同时包含了旋转、移动和缩放等。

6.1.7　同时使用多个变形函数

矩阵变形固然灵活，但是不直观，也不容易理解。transform 属性允许同时使用多个变形函数，这使得元素变形可以更加灵活。示例 6-5 中所实现的效果，也可以通过同时使用多个变形函数来实现。

【示例 6-6】　多种变形的菜单。

下面我们同时使用旋转、移动和缩放来实现如图 6-9 所示的效果。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>多种变形的元素</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
```

```
li:hover {
    background-color:#999;                          /* 深灰色 */
}
a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
}
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：倾斜 */
    -webkit-transform:translate(10px,10px) rotate(30deg) scale(1,-1);
                                                    /* 兼容 webkit 内核 */
    -moz-transform:translate(10px,10px) rotate(30deg) scale(1,-1);
                                                    /* 兼容 gecko 内核 */
    -o-transform:translate(10px,10px) rotate(30deg) scale(1,-1);
                                                    /* 兼容 presto 内核 */
    -ms-transform:translate(10px,10px) rotate(30deg) scale(1,-1);
                                                    /* 兼容 IE9 */
    transform:translate(10px,10px) rotate(30deg) scale(1,-1);
                                                    /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-9 所示。

代码分析：在示例 6-6 中，为了对比变形的差别，仍然保留菜单项变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为 3 个变形函数，分别是移动 translate ()、旋转 rotate()和缩放 scale()。其中 3 个函数执行的顺序依次为：移动、旋转、缩放。示例运行结果如图 6-9 所示，鼠标划过菜单会产生变形。

同样是使用多个变形函数，并给每个函数赋以相同的参数，如果顺序不同，变形的结果也可能不同。调整示例 6-6 中的变形样式如下：

```
<style type="text/css">
…  /* 这里省略的样式不变 */
a:hover {
    background-color:#f90;
    color:#FFF;
    /* 变形方式：倾斜 */
    -webkit-transform: rotate(30deg) translate(10px,10px) scale(1,-1);
                                                        /* 兼容 webkit 内核 */
    -moz-transform: rotate(30deg) translate(10px,10px) scale(1,-1);
                                                        /* 兼容 gecko 内核 */
    -o-transform: rotate(30deg) translate(10px,10px) scale(1,-1);
                                                        /* 兼容 presto 内核 */
```

```
    -ms-transform: rotate(30deg) translate(10px,10px) s cale(1,-1);
                                                  /* 兼容 IE9 */
    transform: rotate(30deg) translate(10px,10px) scale(1,-1);
                                                  /* 标准写法 */
}
</style>
```

运行结果如图 6-9 所示。



图 6-9　运行效果图

代码分析：调整后的样式，仅变形函数的顺序调换了一下，3 个函数的执行顺序变为：旋转、移动、缩放，就产生不一样的变形结果，如图 6-9 所示，与图 6-8 所示的图像在移动方面产生了偏差。因为在旋转执行后，旋转的是该元素对应的坐标系统，即所谓的水平和垂直已经与实际的水平和垂直有了一定的角度，最终移动的结果也不同。

6.1.8　定义变形原点——transform-origin 属性

变形属性 transform 默认的变形原点是元素对象的中心点。CSS 3 提供的 transform-origin 属性，可用于指定这个原点的位置，这个位置可以是元素对象的中心点以外的任意位置，进一步增加了变形的灵活性。

1. 参数说明

定义变形原点的 transform-origin 属性，语法表示如下：

```
transform-origin :<x-axis> <y-axis> ;
```

取值说明如下。

- <x-axis>：定义变形原点的横坐标位置，默认值为 50%，取值包括 left、center、right、百分比值、长度值。
- <y-axis>：定义变形原点的纵坐标位置，默认值为 50%，取值包括 top、middle、bottom、百分比值、长度值。

其中，百分比是相对于元素对象的宽度和高度而言的，而该坐标位置的计算，是以元素的左上角为坐标原点进行计算的。

基于 webkit 内核的替代私有属性是-webkit-transform-origin；基于 gecko 内核的替代私有属性是-moz-transform-origin；基于 presto 内核的替代私有属性是-o-transform-origin；IE 9 浏览器的替代私有属性是-ms-transform-origin。

2. 示例介绍

【示例 6-7】 定义菜单旋转的原点。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>定义菜单旋转的原点</title>
<style type="text/css">
ul {
    margin-top:30px;
    list-style:none;
    line-height:25px;
    font-family:Arial;
    font-weight:bold;
}
li {
    width:120px;
    float:left;
    margin:2px;
    border:1px solid #ccc;
    background-color:#e4e4e4;
    text-align:left;
}
li a {
    display:block;
    padding:5px 10px;
    color:#333;
    text-decoration:none;
    /* 变形原点：自定义 */
    -webkit-transform-origin:0 0;       /* 兼容 webkit 内核 */
    -moz-transform-origin:0 0;          /* 兼容 gecko 内核 */
    -o-transform-origin:0 0;            /* 兼容 presto 内核 */
    -ms-transform-origin:0 0;           /* 兼容 IE9 */
    transform-origin:0 0;               /* 标准写法 */
}
li:hover {
    background-color:#999;              /* 深灰色 */
}
li:hover a{
    background-color:#f90;
    color:#FFF;
    /* 变形方式：旋转 */
    -webkit-transform:rotate(30deg);    /* 兼容 webkit 内核 */
    -moz-transform:rotate(30deg);       /* 兼容 gecko 内核 */
    -o-transform:rotate(30deg);         /* 兼容 presto 内核 */
    -ms-transform:rotate(30deg);        /* 兼容 IE9 */
    transform:rotate(30deg);            /* 标准写法 */
}
</style>
</head>
<body>
<ul>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
</html>
```

运行结果如图 6-10 所示。



图 6-10　鼠标经过时，菜单绕左上角旋转

代码分析：在示例 6-7 中，仍然保留菜单项变形前的显示区域。在鼠标经过的样式中，设置 transform 属性值为旋转变形，同时定义了变形的原点为元素的左上角。示例运行结果如图 6-10 所示，鼠标划过菜单绕左上角旋转一定的角度。

6.1.9　实验室：设计图片画廊

通常的图片预览会整齐地摆放图片，如此精确的对齐排版，也只有电脑才能完成。本节我们就模拟一个图片画廊，随意地摆放，还原一种真实的感觉。

1. 案例简介

本节介绍的案例，是模拟一个图片画廊如图 6-11 所示。在一个限定的区域范围内，摆放 15 张图片，每张图片都有不同程度的旋转，并指定旋转的原点。鼠标经过图片时，图片会以左上角为原点调整至正常的角度并放大显示；鼠标离开后，又会还原为原来的状态。

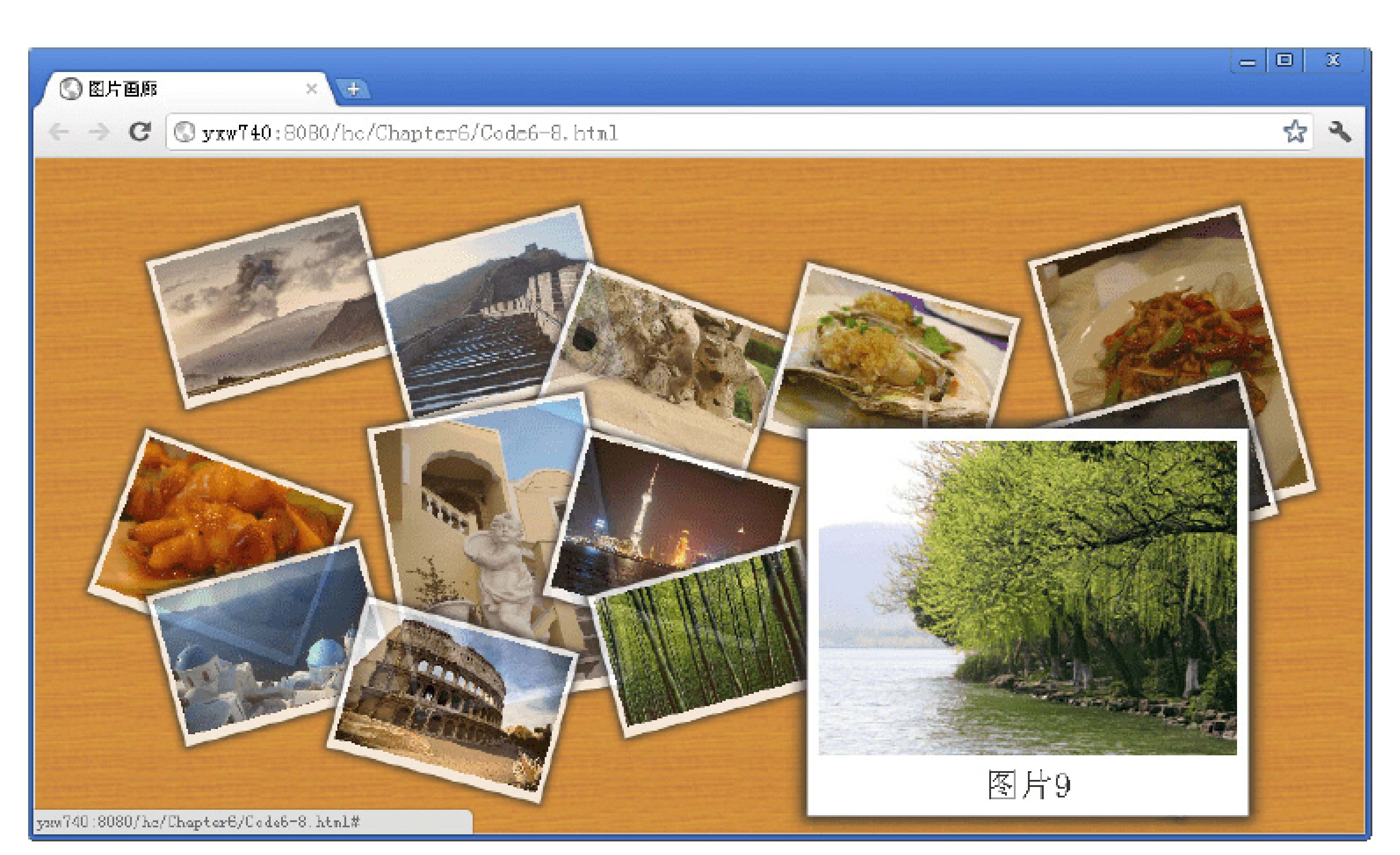


图 6-11　图片画廊

2. 设计网页元素

在网页里以列表的形式添加 15 张图片。

【示例 6-8】 图片画廊。

```
<!DOCTYPEHTML>
<html>
```

```
<head>
<meta charset="utf-8">
<title>图片画廊</title>
</head>
<body>
<ul id="gallery">
  <li><a href="#" title="图片 1"><img src="images/image1.jpg" alt="图片 1"
  /></a></li>
  <li><a href="#" title="图片 2"><img src="images/image2.jpg" alt="图片 2"
  /></a></li>
  <li><a href="#" title="图片 3"><img src="images/image3.jpg" alt="图片 3"
  /></a></li>
  … 省略了相似的代码
  <li><a href="#" title="图片 15"><img src="images/image15.jpg" alt="图片
15" /></a></li>
</ul>
</body>
</html>
```

下面逐步设计样式表，并追加到示例 6-8 中。

3. 设计基本样式表

设置基本的样式表，包括背景墙样式和整体的尺寸布局，链接显示为块级元素，以方便变形和布局。

```
<style type="text/css">
body {
    background:url(images/bark.jpg);       /* 设置背景墙 */
}
#gallery {
    margin: 10px auto;
    padding: 40px;
    list-style:none;
    width:530px;
}
#gallery li {
    float:left;
    width:106px;
    height:80px;
    overflow:visible;                       /* 溢出仍然显示完整内容 */
}
#gallery li a {
    color:#333;
    text-decoration:none;
    font-size:4px;
    display:block;
    text-align:center;
    background-color:#FFF;
    padding:3px;
    opacity:0.8;
    box-shadow:0 0 5px 2px #333;            /* 设置元素阴影 */
}
</style>
```

4. 设计变形样式表

设计出随意摆放的图片效果。首先设置所有的链接默认的倾斜，并自定义变形原点，

同时加入动画过渡效果。使用 CSS 选择器设置不一样的旋转角度。

```
<style type="text/css">
#gallery li a {
    /* 设计变形的过渡效果 */
    -webkit-transition: all 500ms linear;
    -moz-transition: all 500ms linear;
    transition: all 500ms linear;
    /* 自定义变形原点 */
    -webkit-transform-origin:0 0;
    -moz-transform-origin:0 0;
    transform-origin:0 0;
    /* 旋转变形 */
    -webkit-transform:rotate(-15deg);
    -moz-transform:rotate(-15deg);
    transform:rotate(-15deg);
}
#gallery li a img {
    width:100px;
}
#gallery li:nth-child(3n) a {
    -webkit-transform:rotate(20deg);
    -moz-transform:rotate(20deg);
    transform:rotate(20deg);
}
#gallery li:nth-child(4n) a {
    -webkit-transform:rotate(15deg);
    -moz-transform:rotate(15deg);
    transform:rotate(15deg);
}
#gallery li:nth-child(7n) a {
    -webkit-transform:rotate(-10deg);
    -moz-transform:rotate(-10deg);
    transform:rotate(-10deg);
}
#gallery li:nth-child(9n) a {
    -webkit-transform:rotate(-20deg);
    -moz-transform:rotate(-20deg);
    transform:rotate(-20deg);
}
</style>
```

5. 设计鼠标划过时的样式表

设计鼠标划过时，图片调整为正常角度并放大显示，同时也给图片添加说明性的内容。

```
<style type="text/css">
#gallery li a:hover {
    position:relative;
    z-index: 999;
    opacity: 1;
    /* 旋转并放大 */
    -webkit-transform: rotate(0deg) scale(2);
    -moz-transform: rotate(0deg) scale(2);
    transform: rotate(0deg) scale(2);
}
#gallery li a:hover:after {
    content:attr(title);  /* 添加 title 属性内容 */
```

```
}
</style>
```

至此，已经完成了图片画廊的实现。即可以展现如图 6-11 所示的页面效果。

6.2　CSS 3 过渡效果

在 CSS 3 中，transform 属性所实现的元素变形，仅仅呈现的是变形结果。CSS 3 的过渡效果，可以让元素变形看起来比较平滑。本节就对其中的过渡效果进行讲解。

6.2.1　实现过渡效果——transition 属性

CSS 3 新增 transition 属性，可以实现元素变换过程中的过渡效果，即实现了基本的动画。与元素变形属性一起使用，可以展现元素的变形过程，丰富动画的效果。

1. 参数说明

transition 属性用于定义元素变换过程中的过渡效果，语法如下：

```
transition :transition-property || transition-duration || transition-
timing-function || transition-delay ;
```

取值说明如下。

- <transition-property>：定义用于过渡的属性。
- <transition-duration>：定义过渡过程需要的时间。
- <transition-timing-function>：定义过渡方式。
- <transition-delay>：定义开始过渡的延迟时间。

transition 属性定义一组过渡效果，需要上面 4 个方面的参数；transition 属性可以同时定义两组或两组以上的过渡效果，每组用逗号间隔。

基于 webkit 内核的私有属性是-webkit-transition；基于 gecko 内核的私有属性是-moz-transition；基于 presto 内核的替代私有属性是-o-transition。

2. 包含子属性

transition 属性是一个复合属性，可以同时定义过渡效果所需要的参数信息。其包含 4 个方面的信息，就有 4 个子属性：transition-property、transition-duration、transition-timing-function、transition-delay。

对于子属性，基于 webkit 内核的浏览器需增加前缀“-webkit-”，基于 gecko 内核的浏览器需增加前缀“-moz-”，基于 presto 内核的浏览器需增加前缀“-o-”，以使用各种内核的私有属性。

3. 示例介绍

【示例 6-9】 过渡效果。

```
<!DOCTYPEHTML>
<html>
```

```
<head>
<meta charset="utf-8">
<title>过渡效果</title>
<style type="text/css">
div {
    margin:100px auto;
    width:200px;
    height:100px;
    background-color:#00F;
    /* 设置过渡效果 */
    -webkit-transition:all 1000ms linear 500ms;     /* 兼容 webkit 内核 */
    -moz-transition:all 1000ms linear 500ms;        /* 兼容 gecko 内核 */
    -o-transition:all 1000ms linear 500ms;          /* 兼容 presto 内核 */
    transition:all 1000ms linear 500ms;             /* 标准写法 */
}
div:hover  {
    background-color:#F90;
    /* 设置变形：旋转 240deg */
    -webkit-transform:rotate(240deg);
    -moz-transform:rotate(240deg);
    -o-transform:rotate(240deg);
    transform:rotate(240deg);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-12 所示。

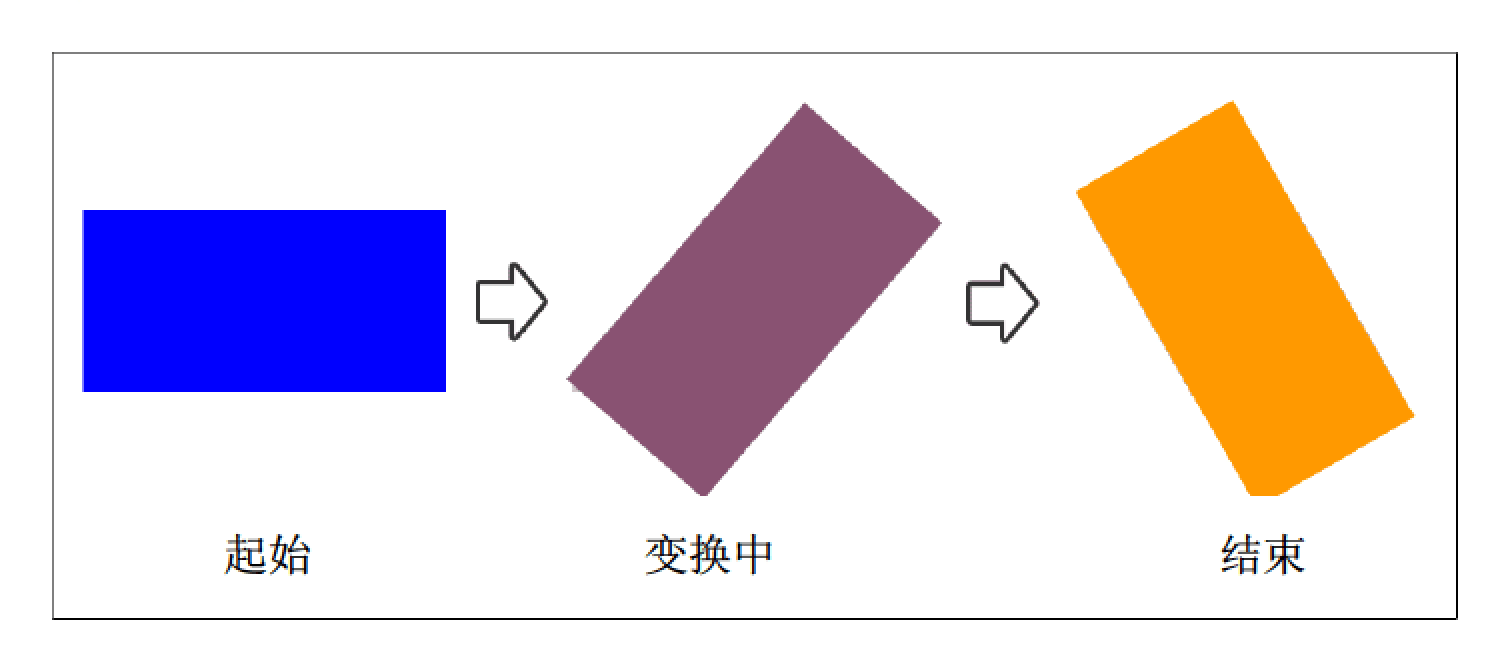


图 6-12　元素变换的过渡效果

代码分析：在示例 6-9 中，为元素的样式变化赋予过渡效果，并在元素的:hover 事件中设置了背景颜色和旋转变形。如图 6-12 所示，左边为起始状态，中间为变换过程中的某个状态，右边为变换结束的状态。有了过渡效果，才算实现了基本的动画效果。

6.2.2　指定过渡的属性——transition-property 属性

1. 参数说明

transition-property 子属性用于定义过渡的属性，语法如下：

```
transition-property : none | all | <property> ;
```

取值说明如下。

- ❑ none：表示没有任何 CSS 属性有过渡效果。
- ❑ all：为默认值，表示所有的 CSS 属性都有过渡效果。
- ❑ <property>：指定一个用逗号分隔的多个属性，针对指定的这些属性有过渡效果。

2. 示例介绍

【示例 6-10】 指定个别属性有过渡效果。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>指定个别属性有过渡效果</title>
<style type="text/css">
div {
    margin:100px auto;
    width:200px;
    height:100px;
    background-color:#00F;
    /* 设置过渡属性 */
    -webkit-transition-property:-webkit-transform;
    -moz-transition-property:-moz-transform;
    -o-transition-property:-o-transform;
    transition-property:transform;
    /* 设置过渡时间 */
    -webkit-transition-duration:1s;
    -moz-transition-duration:1s;
    -o-transition-duration:1s;
    transition-duration:1s;
}
div:hover  {
    background-color:#F90;
    /* 设置变形：旋转 240deg */
    -webkit-transform:rotate(240deg);
    -moz-transform:rotate(240deg);
    -o-transform:rotate(240deg);
    transform:rotate(240deg);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-13 所示。

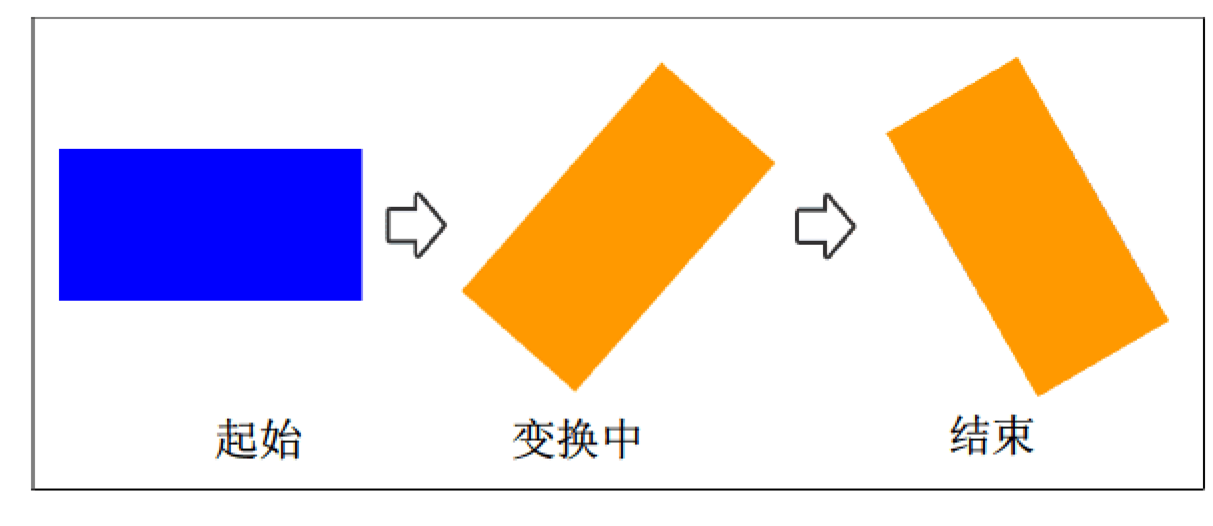


图 6-13　部分属性的过渡效果

代码分析：在示例 6-10 中，使用 transition-property 属性指定了变形属性，没有指定背景属性，所以在变换中的过渡效果，只有变形的过渡，背景的变换没有过渡。如图 6-13 所示，变换中的背景颜色没有过渡效果。

6.2.3　指定过渡的时间——transition-duration 属性

1．参数说明

transition-duration 子属性用于定义过渡过程中需要的时间。语法如下：

```
transition-duration : <time> ;
```

取值说明：<time>指定一个用逗号分隔的多个时间值，时间的单位可以是 s（秒）或 ms（毫秒）。默认情况下为 0，即看不到过渡效果，看到的直接是结果。

2．示例介绍

【示例 6-11】 为过渡属性指定不同的过渡时间。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>过渡效果</title>
<style type="text/css">
div {
    margin:100px auto;
    width:200px;
    height:100px;
    background-color:#00F;
    /* 设置多个过渡属性 */
    -webkit-transition-property:background-color,-webkit-transform;
    -moz-transition-property:background-color,-moz-transform;
    -o-transition-property:background-color,-o-transform;
    transition-property:background-color,transform;
    /* 设置多个过渡时间 */
    -webkit-transition-duration:4s,1s;
    -moz-transition-duration:4s,1s;
    -o-transition-duration:4s,1s;
    transition-duration:4s,1s;
}
div:hover {
    background-color:#F90;
    /* 设置变形：旋转 240deg */
    -webkit-transform:rotate(240deg);
    -moz-transform:rotate(240deg);
    -o-transform:rotate(240deg);
    transform:rotate(240deg);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-14 所示。

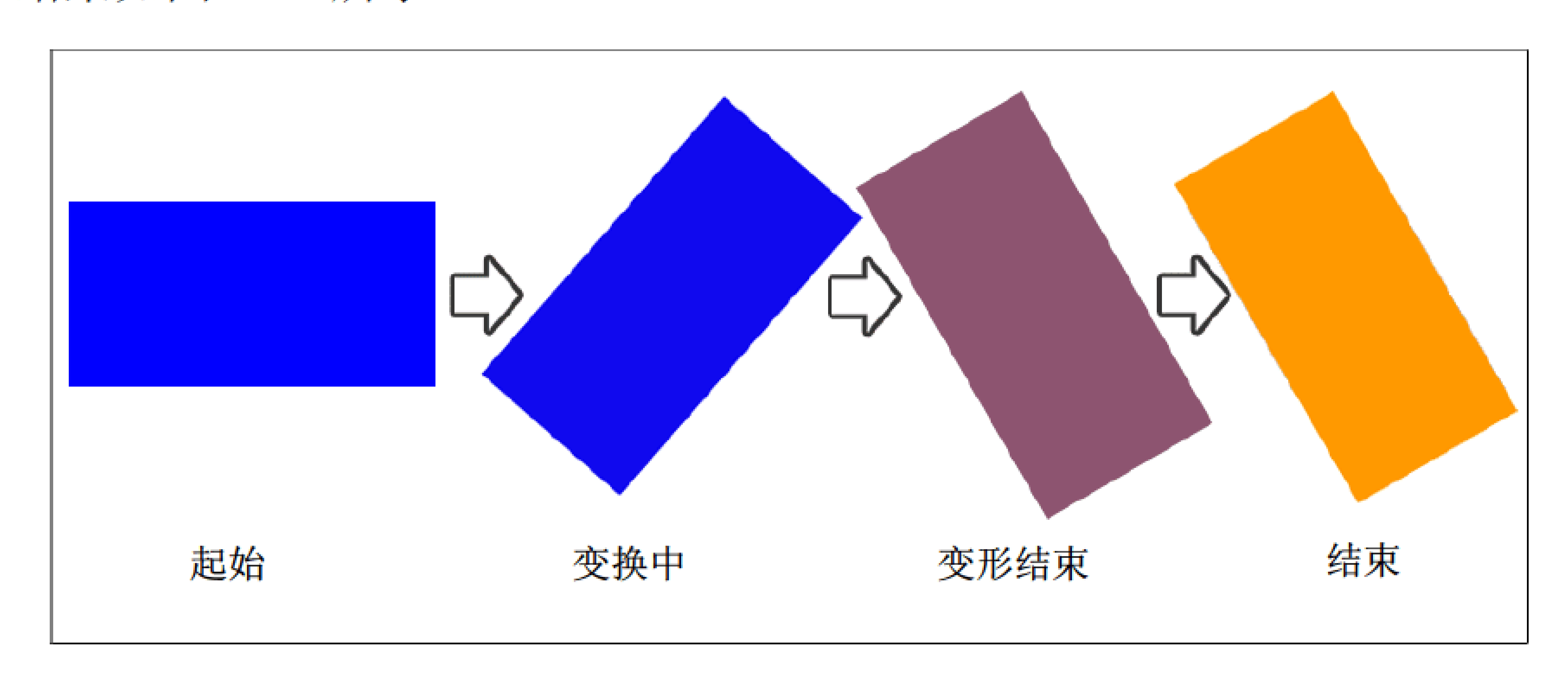


图 6-14　过渡时间不同的过渡效果

代码分析：在示例 6-11 中，指定了两个过渡时间 4s 和 1s，分别应用于背景属性和变形属性。如图 6-14 所示，变形的过渡效果都已经结束了，背景颜色的过渡效果还在持续，直至背景的过渡完成。

6.2.4　指定过渡延迟时间——transition-delay 属性

1．参数说明

transition-delay 子属性用于定义过渡的延迟时间。语法如下：

```
transition-delay :<time> ;
```

取值说明：<time>指定一个用逗号分隔的多个时间值，时间的单位可以是 s（秒）或 ms（毫秒）。默认情况下为 0，即没有时间延迟，立即开始过渡效果。

时间可以为负值，但过渡的效果会从该时间点开始，之前的过渡效果将被截断。

2．示例介绍

【示例 6-12】 延迟的过渡效果。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>过渡效果</title>
<style type="text/css">
div {
    margin:100px auto;
    width:200px;
    height:100px;
    background-color:#00F;
    /* 设置过渡属性 */
    -webkit-transition-property:all;
    -moz-transition-property:all;
    -o-transition-property:all;
    transition-property:all;
    /* 设置过渡时间 */
    -webkit-transition-duration:500ms;
```

```
    -moz-transition-duration:500ms;
    -o-transition-duration:500ms;
    transition-duration:500ms;
    /* 设置延迟时间 */
    -webkit-transition-delay:500ms;
    -moz-transition-delay:500ms;
    -o-transition-delay:500ms;
    transition-delay:500ms;
}
div:hover  {
    background-color:#F90;
    /* 设置变形：旋转 240deg */
    -webkit-transform:rotate(240deg);
    -moz-transform:rotate(240deg);
    -o-transform:rotate(240deg);
    transform:rotate(240deg);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-12 所示。代码分析：在示例 6-12 中，设置了所有属性的改变，都有过渡效果，并设置了过渡效果的延迟时间。当鼠标经过时，需要等待 500ms 后，才产生过渡效果。过渡效果如图 6-12 所示，不同的是，过渡效果延迟了一段时间才开始。

6.2.5　指定过渡方式——transition-timing-function 属性

1. 参数说明

transition-timing-function 子属性用于定义过渡的速度曲线，即过渡方式。语法如下：

```
transition-timing-function :ease | linear | ease-in | ease-out | ease-in-out
| cubic-bezier(n,n,n,n) ;
```

取值说明如下。

- linear：表示过渡一直是一个速度，相当于 cubic-bezier(0,0,1,1)。
- ease：属性的默认值，表示过渡的速度先慢、再快、最后非常慢，相当于 cubic-bezier(0.25,0.1,0.25,1)。
- ease-in：表示过渡的速度先慢、后越来越快，直至结束，相当于 cubic-bezier(0.42,0,1,1)。
- ease-out：表示过渡的速度先快、后越来越慢，直至结束，相当于 cubic-bezier(0,0,0.58,1)。
- ease-in-out：表示过渡的速度在开始和结束时，都很慢，相当于 cubic-bezier(0.42,0,0.58,1)。
- cubic-bezier(n,n,n,n)：自定义贝塞尔曲线效果，其中的 4 个参数为从 0 到 1 的数字。

2. 示例介绍

【示例 6-13】 渐隐的过渡效果。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>过渡效果</title>
<style type="text/css">
div {
    margin:100px auto;
    width:200px;
    height:100px;
    background-color:#00F;
    /* 设置过渡属性 */
    -webkit-transition-property:all;
    -moz-transition-property:all;
    -o-transition-property:all;
    transition-property:all;
    /* 设置过渡时间 */
    -webkit-transition-duration:1000ms;
    -moz-transition-duration:1000ms;
    -o-transition-duration:1000ms;
    transition-duration:1000ms;
    /* 设置过渡方式 */
    -webkit-transition-timing-function:ease-out;
    -moz-transition-timing-function:ease-out;
    -o-transition-timing-function:ease-out;
    transition-timing-function:ease-out;
}
div:hover  {
    background-color:#F90;
    /* 设置变形：旋转 240deg */
    -webkit-transform:rotate(240deg);
    -moz-transform:rotate(240deg);
    -o-transform:rotate(240deg);
    transform:rotate(240deg);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-12 所示。

代码分析：在示例 6-13 中，设置了速度越来越慢的过渡效果。当鼠标经过时，快速产生过渡效果，然后缓慢地结束。这种效果比较常用。

6.2.6　实验室：制作滑动的菜单

菜单是网页的重要组成部分，设计一个良好的菜单是很有必要的。下面我们就用 CSS 3 的过渡效果，来实现一个滑动效果的菜单。

1．案例简介

本节介绍的案例是一个有滑动效果的菜单，当鼠标经过菜单时，菜单会加长，颜色也

会改变，中间的过渡呈现的是快速滑动的效果；当鼠标离开时，菜单还原，则呈现缓慢的滑动效果，如图 6-15 所示。为了版面需要，我们制作了三组效果一样的菜单。

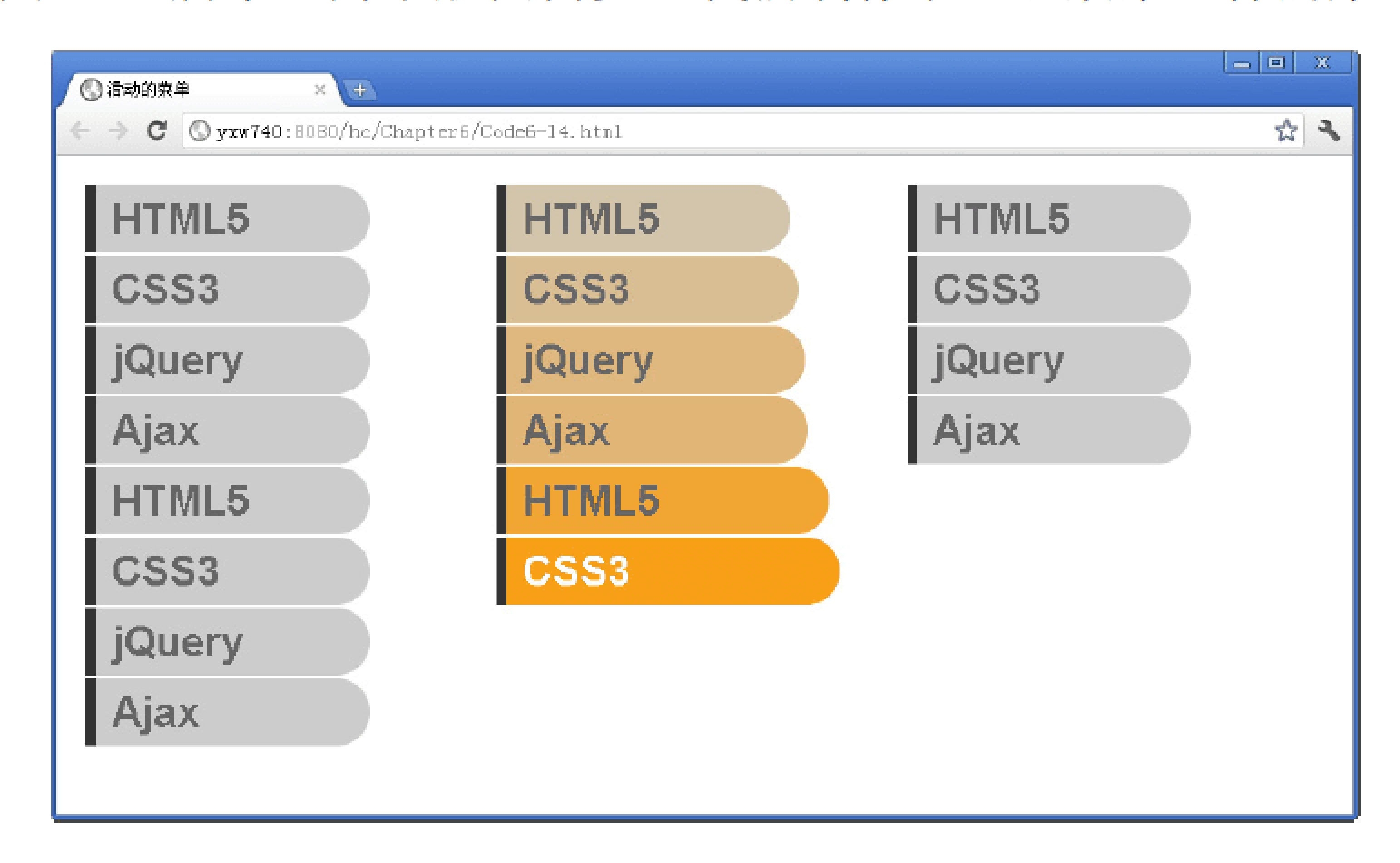


图 6-15　鼠标滑过的菜单

2. 设计网页元素

页面中是三组列表，用于菜单的制作。

【示例 6-14】 滑动的菜单。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>滑动的菜单</title>
</head>
<body>
<ul class="box">
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
<ul class="box">
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
</ul>
<ul class="box">
  <li><a href="#">HTML5</a></li>
  <li><a href="#">CSS 3</a></li>
  <li><a href="#">jQuery</a></li>
  <li><a href="#">Ajax</a></li>
</ul>
</body>
```

```
</html>
```

下面开始进行样式表设计，并追加到示例 6-14 中。

3．设计样式表

大部分是基本的样式表，把每个列表项均设置为菜单形式。其中在列表项中，我们为其设置了时间较长的过渡效果；而在其:hover 事件中设置了时间较短的过渡效果。

```
<style type="text/css">
.box {
    margin:0;
    padding:0;
    font-size:12px;
    list-style:none;
    width:120px;
    float:left;
}
li {
    width:80px;
    line-height:20px;
    height:20px;
    margin:1px;
    background-color:#ccc;
    text-align:left;
    border-radius:0 10px 10px 0;
    border-left:3px solid #333;
    /* 持续时间较长的过渡效果 */
    -webkit-transition:all 1s ease-out;
    -moz-transition:all 1s ease-out;
    -o-transition:all 1s ease-out;
    transition:all 1s ease-out;
}
li a {
    display:block;
    text-decoration:none;
    font-size:12px;
    padding-left:5px;
    font-family:Arial;
    font-weight:bold;
    color:#666;
}
li:hover {
    background-color:#f90;
    width:100px;
    /* 持续时间较短的过渡效果 */
    -webkit-transition:all 200ms linear;
    -moz-transition:all 200ms linear;
    -o-transition:all 200ms linear;
    transition:all 200ms linear;
}
li:hover a {
    color:#FFF;
}
```

```
</style>
```

至此，滑动菜单设计完毕。运行结果如图 6-15 所示，当鼠标经过菜单时，菜单会迅速地过渡到:hover 事件中的效果；当鼠标离开时，菜单状态会缓慢地恢复至初始状态。

6.3　CSS 3 动画设计

前面讲述的元素变形和过渡效果，是制作动画的基础，但还不是真正的动画。本节将学习完整的 CSS 3 动画设计，不但可以创建动画关键帧，还可以对关键帧动画设置播放时间、播放次数、播放方向等，实现更加复杂、更加灵活的动画。

6.3.1　关键帧动画——@keyframes 规则

在动画设计中，关键帧动画是非常重要的功能，它所包含的是一段连续的动画。

1. 参数说明

@keyframes 规则的语法表示如下：

```
@keyframes<animationname> { <keyframes-selector> { <css-styles>}}
```

取值说明如下。

- <animationname>：动画的名称。必须定义一个动画名称，方便与动画属性 animation 绑定。
- <keyframes-selector>：动画持续时间的百分比，也可以是 from 和 to。from 对应的是 0%，to 对应的是 100%，建议使用百分比。必须定义一个，才能实现动画。
- <css-styles>：设置的一个或多个合法的样式属性。必须定义一些样式，才能实现动画。

动画，是通过从一种样式逐步转变到另一种样式来创建的。在指定 CSS 样式变化时，可以从 0%到 100%，逐步设计样式表的变化。

2. 示例介绍

使用动画属性来控制动画的呈现，关键帧动画是通过名称与动画绑定的。

【示例 6-15】 变换位置的小球。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>变换位置的小球</title>
<style type="text/css">
div {
    position:absolute;
    -moz-animation:mymove 5s infinite;      /* mymove 绑定到动画，gecko 内核 */
    -webkit-animation:mymove 5s infinite;    /* mymove 绑定到动画，webkit 内核 */
}
```

```
@-moz-keyframes mymove {          /* 创建关键帧动画，gecko 内核 */
    0% {top:0px;}
    25% {top:200px; left:200px;}
    75% {top:50px; left:10px;}
    100% {top:100px; left:60px;}
}
@-webkit-keyframes mymove {       /* 创建关键帧动画，webkit 内核 */
    0% {top:0px;}
    25% {top:200px; left:200px;}
    75% {top:50px; left:10px;}
    100% {top:100px; left:60px;}
}
</style>
</head>
<body>
<div><img src="images/point.png" width="30" height="30"></div>
</body>
</html>
```

运行结果如图 6-16 所示。

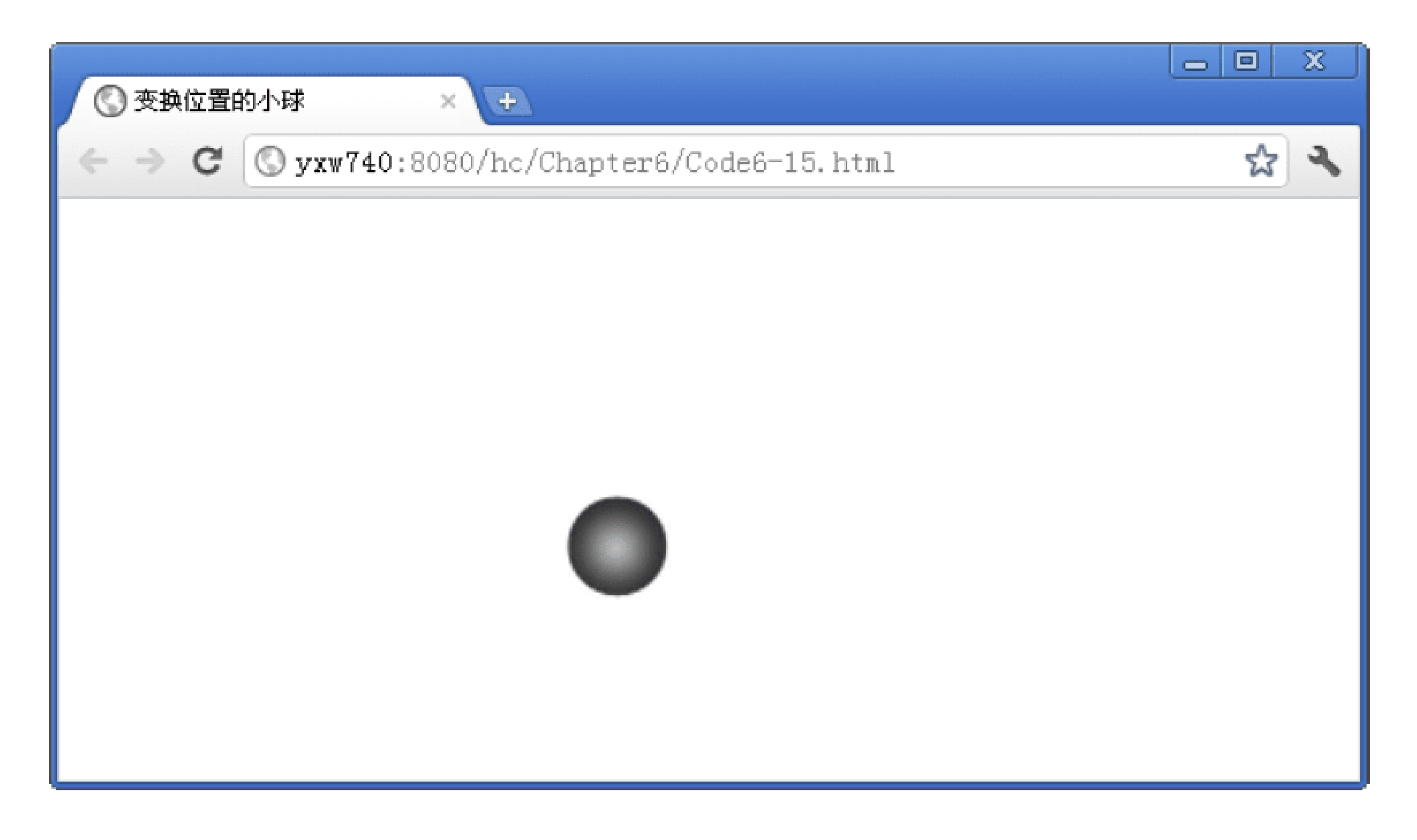


图 6-16　球的位置会不停的变化

代码分析：在示例 6-15 中，创建了名为 mymove 的关键帧动画，并绑定到小球所在的元素中。如图 6-16 所示，小球的位置会发生变化。关于该示例中的 animation 属性将在接下来的章节进行讲解。

6.3.2　动画的实现——animation 属性

CSS 3 提供的 animation 属性，是专门用于动画设计的，它可以把一个或多个关键帧动画绑定到元素上，以实现更加复杂的动画。

1. 参数说明

animation 属性用于同时定义动画所需要的完整信息，其语法如下：

```
animation : <name> <duration> <timing-function> <delay> <iteration-count>
<direction> ;
```

取值说明如下。

- <name>：定义动画的名称，绑定指定的关键帧动画。
- <duration>：定义动画播放的周期时间。
- <timing-function>：定义动画的播放方式，即速度曲线。
- <delay>：定义动画的延迟时间。
- <iteration-count>：定义动画应该播放的次数。
- <direction>：定义动画播放的顺序方向。

animation 属性可以定义一个动画的 6 个方面的参数信息，还可以同时定义多个动画，每个动画的参数信息为一组，用逗号分隔开来。基于 webkit 内核的可替代私有属性是-webkit-animation；基于 gecko 内核的可替代私有属性是-moz-animation。

2．包含子属性

animation 属性是一个复合属性，根据其语法定义，animation 属性包含 6 个子属性：animation-name、animation-duration、animation-timing-function、animation-delay、animation-iteration-count、animation-direction。

对于子属性，基于 webkit 内核的浏览器需增加前缀“-webkit-”，基于 gecko 内核的浏览器需增加前缀“-moz-”，基于 presto 内核的浏览器需增加前缀“-o-”，以使用各种内核的私有属性。

3．指定动画的名称——animation-name属性

animation-name 子属性用来定义动画的名称。该名称是一个动画关键帧名称，是由@keyframes 规则定义的。语法如下：

```
animation-name :<keyframename> | none;
```

取值说明如下。

- none：默认值，表示没有动画。
- <keyframename>：指定动画名称，即指定名称对应的动画关键帧。

如果动画关键帧的名称为 none，则不会显示动画；可以同时指定多个动画名称，多个名称用逗号间隔；如果需要，可以使用 none，取消任何动画。

4．指定动画播放时间——animation-duration属性

animation-duration 子属性用来定义动画播放的周期时间。语法如下：

```
animation-duration : <time>;
```

取值说明：<time>用于指定播放动画的时间长度，单位为 m（秒）或 ms（毫秒）。默认值为 0，表示没有动画。

5．延迟动画播放方式——animation-timing-function属性

animation-timing-function 子属性用来定义动画的播放方式。语法如下：

```
animation-timing-function : ease | linear | ease-in | ease-out | ease-in-out
| cubic-bezier(n,n,n,n) ;
```

取值说明：关于这些取值的说明，完全参阅 transition-timing-function 属性的取值说明。

6．指定动画延迟时间——animation-delay属性

animation-delay 子属性用来定义动画播放的延迟时间，可以定义一个动画延迟一段时间再开始播放。语法如下：

```
animation-delay : <time>;
```

取值说明：<time>用于指定播放动画的时间长度，单位为 m（秒）或 ms（毫秒）。默认值为 0，表示没有时间延迟，直接播放动画。

7．指定动画播放次数——animation-iteration-count属性

animation-iteration-count 子属性用来定义动画循环播放的次数。语法如下：

```
animation-iteration-count : infinite | <n> ;
```

取值说明如下。

- infinite：表示无限次的播放下去。
- <n>：该值为数字，表示循环播放的次数。属性的默认值为 1，表示动画只播放一次。

8．指定动画播放方向——animation-direction属性

animation-direction 子属性用来定义动画循环播放的方向。语法如下：

```
animation-direction : normal | alternate ;
```

取值说明如下。

- normal：为默认值，表示按照关键帧设定的动画方向播放。
- alternate：表示动画的播放方向与上一播放周期相反，第一播放周期还是正常的播放方向。

为方便实现复杂的动画，CSS 3 提供了@keyframes 规则，用于创建动画的关键帧。

9．示例介绍

下面的示例中，我们尝试给元素同时指定多个关键帧动画。

【示例 6-16】 变化的页面元素。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>变化的页面元素</title>
<style type="text/css">
div {
    position:absolute;
    width:100px;
    height:100px;
    top:50px;
    left:100px;
```

```
    background-color:#F90;
    /* 绑定两个关键帧动画：mymove,myrotate */
    -moz-animation-name:mymove,myrotate;
    -webkit-animation-name:mymove,myrotate;
    /* 无限循环 */
    -moz-animation-iteration-count:infinite;
    -webkit-animation-iteration-count:infinite;
    /* 线性变化 */
    -moz-animation-timing-function:linear;
    -webkit-animation-timing-function:linear;
    /* 设置两个关键帧播放时间：4s,3s */
    -moz-animation-duration:4s,3s;
    -webkit-animation-duration:4s,3s;
}
/* 创建关键帧动画 mymove */
 @-moz-keyframes mymove {
    50% {top:50px; left:100px;background-color:#00F;}
}
 @-webkit-keyframes mymove {
    50% {top:150px; left:200px;background-color:#00F;}
}
/* 创建关键帧动画 myrotate */
 @-moz-keyframes myrotate {
    100% {-moz-transform:rotate(360deg);}
}
 @-webkit-keyframes myrotate {
    100% {-webkit-transform:rotate(360deg);}
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-17 所示。



图 6-17　变化的页面元素

代码分析：在示例 6-16 中，定义了两个关键帧动画，分别是移动变换背景的动画 mymove 和旋转变换的动画 myrotate。在 animation-name 属性中，同时指定了这两个关键帧动画，并在 animation-duration 属性中设定了不同的动画播放周期时间。如图 6-17 所示，

页面元素会同时执行两个动画。

6.3.3　实验室：永不停止的风车

如果在网页中使用动画，通常会使用 Flash 动画。如果 CSS 能够帮我们实现动画，是多么令人兴奋的事情啊。本节就用 CSS 3 实现网页中的动画。

1．案例简介

本节介绍的案例是使用 CSS 设计一个风车动画，并让它不停地转动。风车的扇叶和风车杆是由页面元素模拟而成的，案例效果如图 6-18 所示。

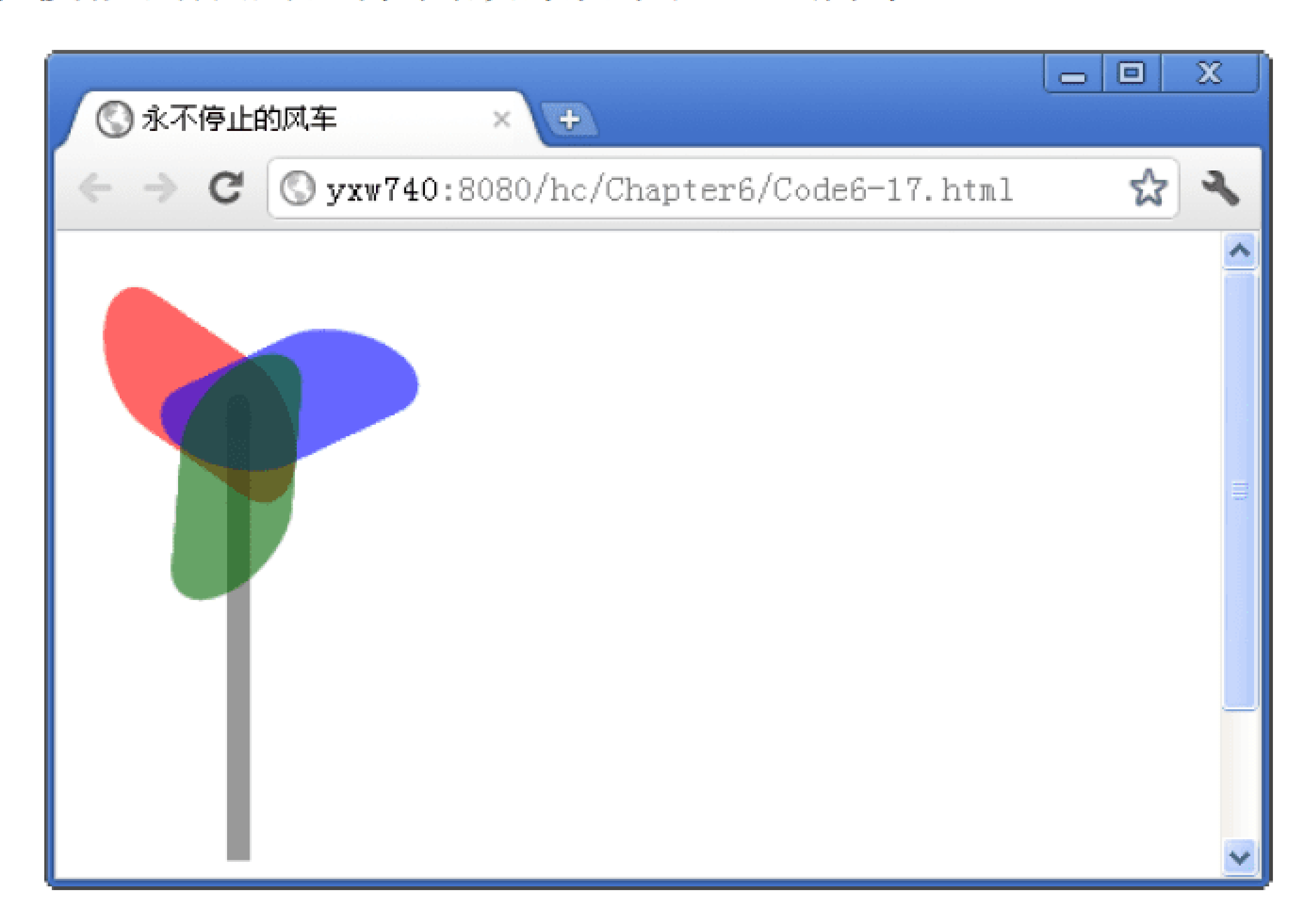


图 6-18　永不停止的风车

2．设计网页元素

风车扇叶分别用三个 span 元素模拟。风车杆中的图片，用于模拟扇叶转动的轴心位置。

【示例 6-17】　永不停止的风车。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>永不停止的风车</title>
</head>
<body>
<!--风车扇叶-->
<div class="pinwheel"> <span class="one"></span>
<span class="tow"></span><span class="three"></span></div>
<!--风车杆-->
<div class="tree"><img src="images/point.png" /></div>
</body>
</html>
```

3．设计风车形状样式表

使用 CSS 变形原理和圆角等模拟风车扇叶；风车杆设计成有渐变效果的。

```
<style type="text/css">
.pinwheel {
    width:140px;
    height:140px;
}
.pinwheel span{
    width:100px;
    height:50px;
    display:block;
    opacity:0.6;
    position:relative;
    border-radius:25px;
}
.pinwheel .one{
    background-color:#f00;
    -webkit-transform:skew(30deg);              /* 倾斜变形 */
    top:48px;
    left:38px;
}
.pinwheel .tow{
    background-color:#00f;
    -webkit-transform: rotate(120deg) skew(30deg);
                                                /* 旋转 120deg，并倾斜变形 */
    top:18px;
    left:5px;
}
.pinwheel .three{
    background-color:#060;
    -webkit-transform: rotate(240deg) skew(30deg);
                                                /* 旋转 240deg，并倾斜变形 */
    top:-72px;
    left:5px;
}
.pinwheel .point{
    position:relative;
    top:-90px;
    left:45px;
}
.tree{
    position:relative;
    top:-78px;
    left:65px;
    border-radius:10px 10px 0 0;
    height:200px;
    width:10px;
    background-color:#999;
    z-index:-1;
}
.tree img{
    width:10px;
}
</style>
```

4．设计风车动画

创建关键帧动画 keyname，并绑定到元素的动画中。

```
<style type="text/css">
.pinwheel {
```

```
    -webkit-transform-origin:69px 73px; /* 指定风车扇叶变形的中心原点 */
    -webkit-animation-name:keyname;     /* 绑定关键帧动画 */
    -webkit-animation-duration:2s;                 /* 动画播放周期 2s */
    -webkit-animation-iteration-count:infinite;    /* 动画无限制循环 */
    -webkit-animation-timing-function:linear;      /* 线性的变化速度 */
}
@-webkit-keyframes keyname {
    from {
        -webkit-transform:rotate(0);
    }
    to {
        -webkit-transform:rotate(360deg);          /* 旋转风车 360deg */
    }
}
</style>
```

至此，动画设计完成，运行结果如图 6-18 所示，风车会不停地转动。

6.4　CSS 3 渐变设计

渐变，是网页设计中使用频率较高的一种效果，它可以让元素看起来更有质感。传统的渐变实现方式是非常依赖图片的，而 CSS 3 能方便地实现元素的渐变，避免了过多地使用渐变图片所带来的麻烦，而且在放大网页的情况下一样过渡自然。

渐变分两种：线性渐变和径向渐变。遗憾的是，渐变的实现方式还没有统一的标准，各主流浏览器均提供了私有的实现。下面分别对基于各种内核的实现进行讲解。

6.4.1　CSS 线性渐变

1．基于Webkit内核的实现

基于 Webkit 内核的线性渐变，语法如下：

```
-webkit-gradient ( linear,<point>,<point>, from(<color>), to(<color>)[,
color-stop(<percent>,<color>)]* )
```

取值说明如下。

- linear：表示线性渐变类型。
- <point>：定义渐变的起始点和结束点：第一个表示起始点，第二个表示结束点。该坐标点的取值，支持数值、百分比和关键字，如(0.5,0.5)、(50%,50%)、(left,top)等。关键字包括：定义横坐标的 left 和 right，定义纵坐标的 top 和 bottom。
- <color>：表示任意 CSS 颜色值。
- <percent>：表示百分比值，用于确定起始点和结束点之间的某个位置。
- from()：定义起始点的颜色。
- to()：定义结束点的颜色。
- color-stop()：可选函数，在渐变中多次添加过渡颜色，可以实现多种颜色的渐变。

基于 Webkit 内核的渐变语法比较严谨。

2．基于Gecko内核的实现

基于 Gecko 内核的线性渐变，语法如下：

```
-moz-linear-gradient ( [ <point> || <angle>,]?<color>[, <color>
[<percent>]?]*,<color> )
```

取值说明如下。

- <point>：定义渐变的起始点。该坐标的取值支持数值、百分比和关键字。关键字包括：定义横坐标的 left、center 和 right，定义纵坐标的 top、center 和 bottom。默认坐标为（top center）。当指定一个值时，另一个值默认为 center。
- <angle>：定义线性渐变的角度。单位可以是 deg（角度）、grad（梯度）、rad（弧度）。
- <color>：表示渐变使用的 CSS 颜色值。
- <percent>：表示百分比值，用于确定起始点和结束点之间的某个位置。

这里没有函数作为参数，可以直接在某个百分比位置添加过渡颜色。第一个颜色值为渐变开始的颜色，最后一个颜色值为渐变结束的颜色。基于 Gecko 内核的线性渐变的实现，比较符合 W3C 语法标准。

3．示例介绍

下面结合示例来演示线性渐变的实现方法。

【示例 6-18】 线性渐变的背景。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>线性渐变的背景</title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    background-color:#F90;
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear,left top,left bottom, from(#f90),
    to(#0f0));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(left,#f90,#0f0);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-19 所示。

代码分析：在示例 6-18 中，设计的是一个从上至下的渐变，实现了基于 Webkit 和 Gecko 两种浏览器的线性渐变。其中基于 Gecko 内核的渐变实现，应用了其默认的设置：当不设

置起点和弧度方向时，默认的是从上至下的渐变。如图 6-19 所示，是在 Chrome 和 Firefox 两种浏览器中运行的效果。



图 6-19　从上到下的线性渐变

再来实现从左至右的线性渐变，调整样式表如下：

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    background-color:#F90;
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear,left top,right top, from(#f90),
    to(#0f0));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(top,#f90,#0f0);
}
</style>
```

运行结果如图 6-20 所示。



图 6-20　从左至右的线性渐变

再来实现从左上角到右下角的渐变，调整样式表如下：

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    background-color:#F90;
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear,left top,right bottom, from(#f90),
    to(#0f0));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(left top,#f90,#0f0);
}
</style>
```

运行结果如图 6-21 所示。



图 6-21　从左上角至右下角的线性渐变

再来实现在渐变中增加过渡颜色。调整样式表如下：

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    background-color:#F90;
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear,left top,right top, from(#f90),
    to(#0f0),color-stop(50%,blue));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(left,#f90,blue,#0f0);
}
</style>
```

运行结果如图 6-22 所示。

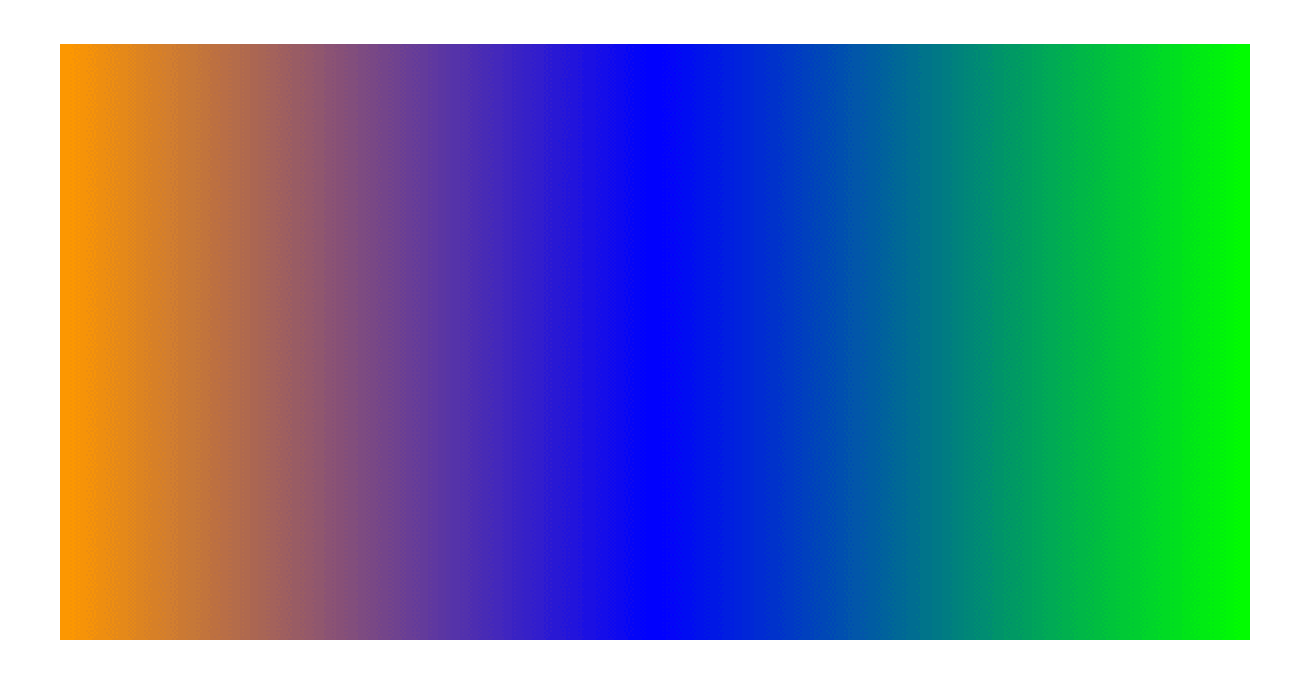


图 6-22　多种颜色的线性渐变

从上面的这些示例可以看出，基于 Gecko 内核的渐变实现比较简单，但不易理解；基于 Webkit 内核的渐变实现代码较长，但逻辑层次比较清晰。基于 IE 内核的渐变，是借助滤镜实现的，这里不再讲解。

6.4.2　CSS 径向渐变

1．基于Webkit内核的实现

基于 Webkit 内核的径向渐变，语法如下：

```
-webkit-gradient ( radial [,<point>,<radius>]{2}, from(<color>),
to(<color>)[,color-stop(<percent>,<color>)]* )
```

取值说明如下。

- radial：表示径向渐变类型。
- <point>：定义渐变的起始圆的圆心坐标和结束圆的圆心坐标。该坐标点的取值，支持数值、百分比和关键字，如(0.5,0.5)、(50%,50%)、(left,top)等。关键字包括：定义横坐标的 left 和 right，定义纵坐标的 top 和 bottom。
- <radius>：表示圆的半径，定义起始圆的半径和结束圆的半径。默认为元素尺寸的一半。
- <color>：表示任意 CSS 颜色值。
- <percent>：表示百分比值，用于确定起始点和结束点之间的某个位置。
- from()：定义起始圆的颜色。
- to()：定义结束圆的颜色。
- color-stop()：可选函数，在渐变中多次添加过渡颜色，可以实现多种颜色的渐变。

2．基于Gecko内核的实现

基于 Gecko 内核的径向渐变，语法如下：

```
-moz-radial-gradient ( [ <point> || <angle>,]?[<shape>||<radius>]?<color>[,
<color> [<percent>]?]*,<color> )
```

取值说明如下。

- <point>：定义渐变的起始点。该坐标的取值支持数值、百分比和关键字。关键字包括：定义横坐标的 left、center 和 right，定义纵坐标的 top、center 和 bottom。默认坐标为（center center）。当指定一个值时，另一个值默认为 center。
- <angle>：定义径向渐变的角度。单位可以是 deg（角度）、grad（梯度）、rad（弧度）。
- <shape>：定义径向渐变的形状，包括 circle（圆形）和 ellipse（椭圆形）。默认为 ellipse。
- <radius>：定义圆的半径或者椭圆的轴长度。
- <color>：表示渐变使用的 CSS 颜色值。
- <percent>：表示百分比值，用于确定起始圆和结束圆之间的某个位置。

这里没有函数作为参数，可以直接在某个百分比位置添加过渡颜色。第一个颜色值为渐变开始的颜色，最后一个颜色值为渐变结束的颜色。基于 Gecko 内核的径向渐变的实现，比较符合 W3C 语法标准。

3．示例介绍

下面结合示例来演示径向渐变的实现方法。

【示例 6-19】 径向渐变的背景圆。

```
<!DOCTYPEHTML>
<html>
<head>
```

```
<meta charset="utf-8">
<title>径向渐变</title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    background-color:#F90;
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(radial,200 100,10,200 100, 100,
            from(#f90),to(#0f0),color-stop(50%,blue));
    /* 基于 Gecko 内核的实现 */
    background:-moz-radial-gradient(200px 100px,circle,#f90 10px,blue,
    #0f0 100px);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

运行结果如图 6-23 所示。

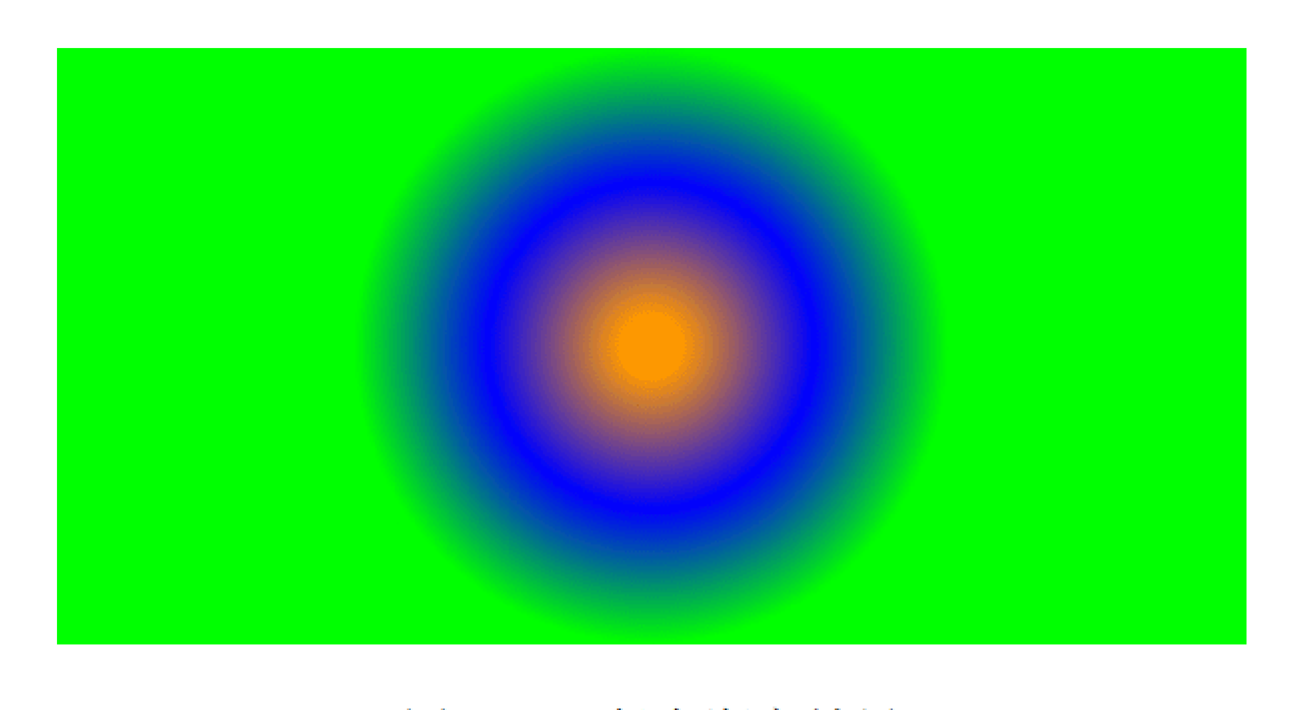


图 6-23　径向渐变效果

代码分析：在示例 6-19 中，设计的是一个径向渐变，实现了基于 Webkit 和 Gecko 两种浏览器径向渐变的实现。可以理解为从一个小圆到一个大圆的渐变。在渐变过程中，增加了蓝色作为过渡颜色。如图 6-23 所示，是在 Chrome 和 Firefox 两种浏览器中运行的效果。

由于基于 Webkit 和 Gecko 的径向渐变实现方法不同，复杂的渐变，很难同时实现。例如，使用基于 Webkit 的-webkit-gradient()，可以轻松实现放射效果；基于 Gecko 的-moz-radial-gradient()，则可以轻松实现椭圆效果。正因为这些无法统一的问题存在，径向渐变在实际使用过程中比较受限制。复杂的径向渐变，这里不再讲述。

6.4.3　实验室：设计渐变的按钮

在页面设计中，渐变的应用随处可见，适当地使用渐变，可使网页更具层次性。使用渐变最好的地方，就是按钮了。下面我们就使用渐变设计不同形状的按钮。由于按钮风格是 CSS 样式设计的，因此也可以应用在其他页面元素中。

1．案例简介

本节介绍的案例是使用 CSS 渐变设计一组常用的按钮样式。这些样式可以应用在任何 HTML 元素中。示例中根据圆角的半径设置三种风格，背景为线性渐变，效果图如图 6-24 所示。



图 6-24　渐变的按钮

2．设计网页元素

设计三组按钮样式，准备三组同样的页面元素标签，每组标签均为 span、a、input 三个元素。下面的样式表，会把这些标签设计成按钮风格。

【示例 6-20】 设计渐变的按钮。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>设计渐变的按钮</title>
</head>
<body>
<div> <span class="button">Span</span>
  <input type="button" value="Buttom" class="button" />
  <a href="#" class="button">Link</a> </div>
<div> <span class="button radius5">Span</span>
  <input type="button" value="Buttom" class="button radius5" />
  <a href="#" class="button radius5">Link</a> </div>
<div> <span class="button radius15">Span</span>
  <input type="button" value="Buttom" class="button radius15" />
  <a href="#" class="button radius15">Link</a> </div>
</body>
</html>
```

3．设计按钮的基本风格

设计按钮的内部布局、阴影效果和圆角效果。

```
<style type="text/css">
div{
    margin-top:10px;
}
.button {
    display:inline;
```

```
    padding:5px 20px;
    font-size:12px;
    font-weight:lighter;
    font-family:Arial, Helvetica, sans-serif;
    border:none;
    color:#333;
    letter-spacing:1px;
    text-decoration:none;
    /* 设置阴影效果 */
    -webkit-box-shadow:2px 2px 2px #333;
    -moz-box-shadow:1px 1px 1px #ccc;
    box-shadow:1px 1px 2px #333;
}
.radius5 {
    /* 圆角半径为 5px */
    -webkit-border-radius:5px;
    -moz-border-radius:5px;
    border-radius:5px;
}
.radius15 {
    /* 圆角半径为 15px */
    -webkit-border-radius:15px;
    -moz-border-radius:15px;
    border-radius:15px;
}
</style>
```

4．设计按钮的渐变效果

设计按钮的渐变效果和鼠标经过时的渐变效果。

```
<style type="text/css">
.button {
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear, left top, left bottom, from
    (#ffcc33), to(#f90));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(#ffcc33, #f90);
}
.button:hover {
    /* 基于 Webkit 内核的实现 */
    background:-webkit-gradient(linear, left top, left bottom, from
    (#ff9933), to(#ff3300));
    /* 基于 Gecko 内核的实现 */
    background:-moz-linear-gradient(#ff9933, #ff3300);
}
</style>
```

至此，渐变风格的按钮设计完成，运行结果如图 6-24 所示。使用时，可根据页面的风格需要，调整渐变的色调。

6.5　小　　结

本章主要讲解了 CSS 的元素变形、过渡效果、动画设计和渐变设计。重点讲解了元素

的各种变形方法、过渡效果的实现，以及如何使用渐变等。比较难以理解的是过渡效果与动画之间的区别，而渐变的语法的语法规则也是本章的难点，望读者多加琢磨、练习。下一章将介绍如何设计支持多种设备的样式表方案。

6.6　习　　题

【习题 1】transform 属性的值是一个变形函数，请问有哪些变形函数？

【习题 2】变形属性 transform 默认的变形原点是元素对象的中心点。如果要改变变形原点，应该使用什么属性？

【习题 3】transition 属性是用来实现过渡效果的，请列举它有哪些子属性。

【习题 4】请描述一下 CSS 3 中的过渡效果与动画之间的区别。

【习题 5】CSS 3 包含两种渐变：线性渐变和径向渐变。请设计一个包含渐变的页面，其中的渐变使用 CSS 3 中的渐变来实现。

第 7 章　支持多种设备的样式表方案

伴随移动互联网的到来，越来越多的人开始使用手机等手持终端设备上网，终端屏幕尺寸也趋于多元化。CSS 3 迎合了这种趋势，提出了媒体查询的新概念，使得设计的样式表在多种终端设备下，都能够很好地呈现网页。本章将详细讲解面向多种媒体的样式表设计。

7.1　媒 体 查 询

在 CSS 中，与媒体相关的样式表设计是从 CSS 2.1 开始的。在 CSS 2.1 中，可以通过 Media Type 来区别终端设备，以指定不同的样式表。例如，需要打印网页时，设计针对打印的样式表。Media Type 极不灵活，而且不曾被多少设备支持。

CSS 3 新增了 Media Queries（媒体查询）模块。该模块中，允许添加媒体查询表达式，以指定媒体类型及设备特性，从而精确地为不同的设备应用不同的样式，最终改善用户体验。

7.1.1　@media 规则的语法

Media Queries 模块中的媒体查询是使用@media 规则来区别媒体设备，并实现样式表定义的。Media Queries 模块以获得了 Firefox、Safari、Chrome 和 Opera 等浏览器的支持。

1．参数说明

@media 规则是包含有查询表达式的媒体样式表定义规则。语法如下：

```
@media <media query> { <css styles>}
<media_query>: [only | not]? <media_type> [ and <expression> ]*
<expression>:  ( <media_feature> [: <value>]? )
<media_type>: all | aural | braille | handheld | print | projection | screen
| tty | tv | embossed
<media_feature>: width | min-width | max-width | height | min-height |
max-height
  | device-width | min-device-width | max-device-width
  | device-height | min-device-height | max-device-height
  |device-aspect-ratio|min-device-aspect-ratio|max-device-aspect-ratio
  | color | min-color | max-color | color-index | min-color-index |
 max-color-index
  | monochrome | min-monochrome | max-monochrome
  | resolution | min-resolution | max-resolution | scan | grid
```

取值说明如下。

- <css_style>：定义样式表。
- <media_query>：设置媒体查询关键字，如 and（逻辑与）、not（排除某种设备）、only（限定某种设备）。
- <media_type>：设置设备类型，语法中提供了 10 种媒体类型，详情如表 7.1 所示。
- <media_feature>：定义媒体特性，该特性放在括号中，如（man-width:800px）。媒体特性有 13 种，详情如表 7.2 所示。

表 7.1　Media Queries媒体类型说明

媒体类型值	媒体类型说明
all	所有设备
screem	电脑显示器
print	用于打印机或打印预览视图
handheld	便携或手持设备
tv	电视机类型的设备
speech	语音和音频合成器
braille	用于盲人触觉反馈设备
embossed	盲文打印/印刷设备
projection	各种投影设备
tty	用于使用固定间距字符格的设备，如电传打字机和终端

表 7.2　Media Queries媒体特性说明

媒体特性	值	可用媒体类型	接受 min/max
width	length	visual、tactile	yes
height	length	visual、tactile	yes
device-width	length	visual、tactile	yes
device-height	length	visual、tactile	yes
orientation	portrait | landscape	bitmap	no
aspect-ratio	ratio	bitmap	yes
device-aspect-ratio	ratio	bitmap	yes
color	integer	visual	yes
color-index	integer	visual	yes
monochrome	integer	visual	yes
resolution	resolution	bitmap	yes
scan	progressive | interlace	tv	no
grid	integer	visual、tactile	no

媒体特性共有 13 种，形式上与 CSS 属性类似。但与 CSS 属性不同的是，大部分设备的指定值接受 min/max 前缀，用来表示大于等于或小于等于的逻辑。

2．示例介绍

下面的示例是根据媒体查询规则，当浏览器窗口大小改变时，会自动选择相应的样式表。

【示例 7-1】 根据窗口尺寸选择不同的样式。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>根据窗口尺寸选择不同的样式</title>
<style type="text/css">
* {
    font-size:36px;
    font-weight:bold;
    font-family:Arial, Helvetica, sans-serif;
    color:#FFF;
}
nav {
    background-color:#0066ff;
    height:300px;
}
section {
    background-color:#f90;
    height:300px;
}
aside {
    background-color:#009900;
    height:300px;
}
/* 窗口宽度大于 900px */
@media screen and (min-width:900px) {
nav {
 float:left;
 width:25%;
}
section {
 float:left;
 width:50%;
}
aside {
 float:left;
 width:25%;
}
}
/* 窗口宽度在 600px 和 900px 之间 */
@media screen and (min-width:600px) and (max-width:900px) {
nav {
 float:left;
 width:40%;
 height:200px;
}
section {
 float:left;
 width:60%;
 height:200px;
}
aside {
 height:100px;
 float:none;
 clear:both;
}
}
/* 窗口宽度小于 600px */
@media screen and (max-width:600px) {
nav {
```

```
 height:150px;
}
section {
 height:150px;
}
aside {
height:150px;
}
}
</style>
</head>
<body>
<nav> Nav </nav>
<section>Section</section>
<aside>Aside</aside>
</body>
</html>
```

运行结果如图 7-1～图 7-3 所示。



图 7-1　窗口宽度大于 900px 时的效果

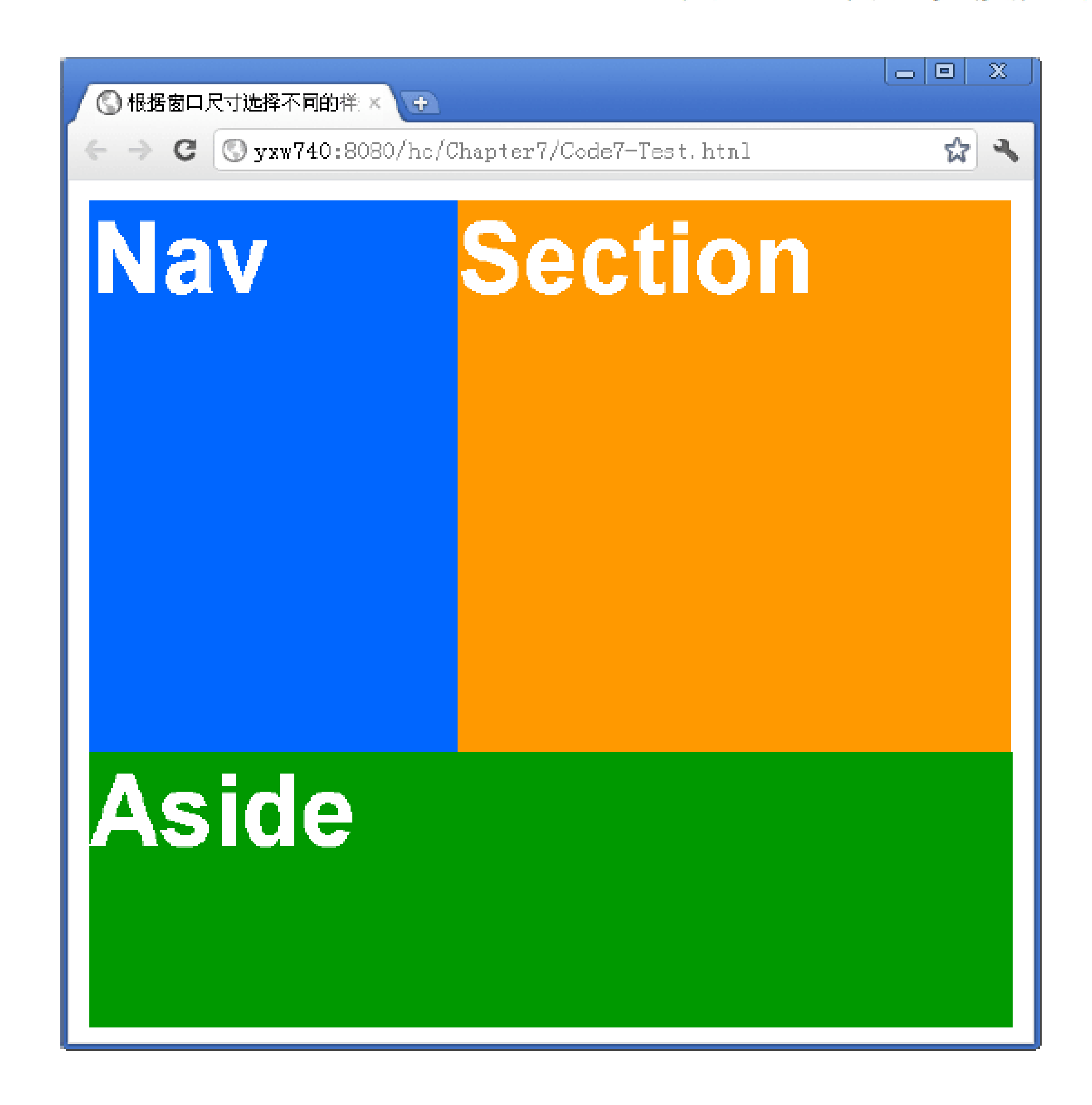


图 7-2　窗口宽度在 600px 和 900px 之间时的效果

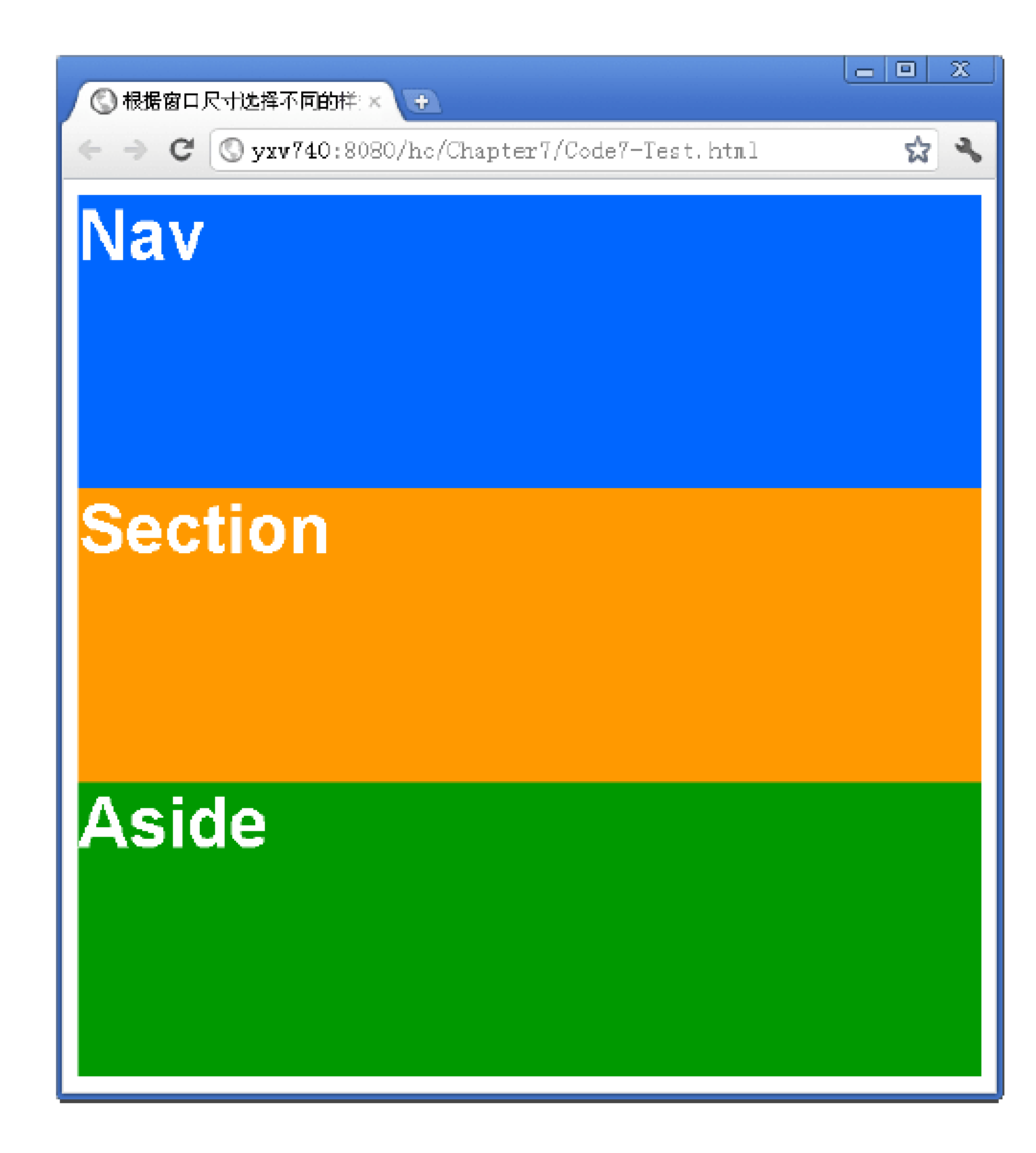


图 7-3　窗口宽度小于 600px 时的效果

代码分析：在示例 7-1 中，使用媒体查询的方法，针对不同的窗口大小，定义了不同的样式表。当窗口宽度超过 900px 时，网页运行效果如图 7-1 所示；当窗口宽度在 600px 和 900px 之间时，网页运行效果如图 7-2 所示；当窗口宽度在 600px 以下时，网页运行效果如图 7-3 所示。

7.1.2　使用 Media Queries 链接外部样式表文件

在网页设计中，通常是直接链接外部样式表文件的，此时也可以增加 Media Queries 媒体查询，以适应媒体设备的需求。

1．在<link>标签中应用Media Queries

在<link>标签中设置 media 属性语法如下：

```
<link rel="stylesheet" type="text/css" media="<media_query>" href=
"xxx.css" />
```

取值说明：<media_query>，媒体查询，遵循@media 规则中的媒体查询方式。

2．示例介绍

一个 Media Query 语句包含一种媒体类型，如果没有指定媒体类型，那么就使用默认类型 all。例如：

```
<link rel="stylesheet" type="text/css" href="xxx.css" media=
"(min-width:200px)" />
```

一个 Media Query 语句也可以同时包含多个特性，例如：

```
<link rel="stylesheet" type="text/css" href="xxx.css"
    media="screen and (min-width:200px) and (max-width:400px)" />
```

一个 Media Query 语句也可以同时包含多个媒体查询，多个查询用逗号隔开，例如：

```
<link rel="stylesheet" type="text/css" href="xxx.css"
    media="handheld and (max-width:200px), screen and (max-width:300px)" />
```

使用 not 关键字，来排除符合表达式的设备，例如：

```
<link rel="stylesheet" type="text/css" href="xxx.css" media="not screen
and (color)" />
```

使用 only 关键字，支持 Media Queries 的设备会正确地显示样式；不支持 Media Query，但能正确读取 Media Type 的设备，由于先读取到 only 而不是 screen 等，将会忽略后面的样式。对于不支持 Media Query 的 IE 来说，无论是否有 only，都会忽略样式。

7.2　实验室：自适应屏幕的样式表方案

使用媒体查询 Media Queries，可以感知屏幕的尺寸，这样就可以为不同尺寸的屏幕设计不同的样式布局。本章就借助媒体查询来实现一个自适应屏幕的页面布局。

1．案例简介

本节介绍的案例是一个简易的网页，以北国风光为主题，页面内容包含一个 logo、一个导航栏和一个图片的列表。本案例中，我们将借助媒体查询，将针对以下屏幕宽度进行设计：大于 900px、小于 900px 且大于 600px、小于 600px 且大于 400px、小于 400px。

为了显示良好，每种屏幕宽度的页面布局也会有相应的调整和改变。案例效果如图 7-4 所示。下面来设计这个网页。



图 7-4　自适应屏幕的网页

2．网页中基本的元素

页面内容包括 logo、导航栏、图片列表和一个底部区域。

【示例 7-2】 自适应屏幕的样式表方案。

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="utf-8">
<title>北国风光</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<!-- 包含 logo 的导航栏 -->
<nav>
  <!-- logo -->
  <h1 id="logo"><a href="#"><img src="images/logo.jpg" alt="北国风光" >
</a></h1>
  <!-- 导航栏 -->
  <ul>
    <li><a href="#">名词来历</a></li>
    <li><a href="#">北国雾凇</a></li>
    <li><a href="#">风光图片集</a></li>
  </ul>
</nav>
<section>
  <!-- 图片列表 -->
  <article>
    <h2 style="margin-top:-15px;"><span>风光图片集</span></h2>
    <ol>
      <li> <a href="#"> <img src="images/Scene1.jpg" alt=""> <span>图片
      1</span> </a> </li>
      <li> <a href="#"> <img src="images/Scene2.jpg" alt=""> <span>图片
      2</span> </a> </li>
      <li> <a href="#"> <img src="images/Scene3.jpg" alt=""> <span>图片
      3</span> </a> </li>
      <li> <a href="#"> <img src="images/Scene4.jpg" alt=""> <span>图片
      4</span> </a> </li>
      <li> <a href="#"> <img src="images/Scene5.jpg" alt=""> <span>图片
      5</span> </a> </li>
      <li> <a href="#"> <img src="images/Scene6.jpg" alt=""> <span>图片
      6</span> </a> </li>
    </ol>
    <div class="clear"></div>
  </article>
  <!-- 页面底部 -->
  <footer> 北国风光&copy; 2011</footer>
</section>
</body>
</html>
```

3. 添加基本的样式表

基本的样式表包括基本的风格设置，暂时不涉及布局。后面我们将会根据媒体查询的结果，设置不同的布局样式表。

```
<style type="text/css" media="screen, projection">
body {
    line-height:1;
    color:#333;
}
ol, ul, h1 {
    margin:0;
    padding:0;
```

```
    list-style:none;
}
a {
    color: #933;
    text-decoration: none;
}
a:hover {
    color: #DF3030;
}
nav h1 {
    text-align:center;
}
nav h1 img {
    width:90%;
}
nav ul {
    border-top: 1px solid #999;
}
nav li {
    text-align: center;
    border-bottom:1px solid #ccc;
    line-height:60px;
}
nav li a {
    display:block;
}
nav li a:hover {
    background-color:#e4e4e4;
}
section {
    font-size:14px;
    font-family:"宋体";
}
section h2 {
    font-size:18px;
    text-align:center;
    font-family:"黑体";
    font-weight:lighter;
}
section span {
    padding:0 10px;
    background-color:#FFF;
}

section li {
    text-align:center;
}
section li img {
    width:98%;
    border-radius:5px;
}
section article {
    border-top: 1px solid #999;
    margin-top:20px;
    padding-bottom:20px;
}
.clear {
    clear:both;
    line-height:5px;
}
footer {
```

```
    clear:both;border-top: 1px solid #999;
    font-size: 12px;
    text-align: center;
    padding: 10px 0;
    font-family:Arial, Helvetica, sans-serif;
    color:#666;
}
</style>
```

4．设计屏幕宽度大于900px的样式表

屏幕宽度大于 900px 时，logo 暂居左侧，导航栏在右侧的顶部，右侧中间是图片列表，右侧下方是底部区域。其中，图片列表每行显示三个图片。风格如图 7-4 中编号 1 的效果所示。样式表设计如下：

```
<style type="text/css" media="screen, projection">
@media (min-width: 900px) {
  nav h1 {
    float:left;
    width:35%;
    height:200px;
}
nav ul {
    float:left;
    width: 65%;
}
nav li {
    float:left;
    width:32%;
    margin-left:1%;
}
section {
    float:left;
    width: 65%;
    padding:20px 0;
}
section li {
    float:left;
    margin:10px 2px;
    width:32%;
}
section li span {
    display:block;
    text-align:center;
    font-size:12px;
}
}
</style>
```

5．设计屏幕宽度小于900px且大于600px的样式表

屏幕宽度小于 900px 且大于 600px 时，logo 和导航栏都会布局在左侧，右侧的上部是图片列表，右侧下方是底部区域。其中，图片列表每行显示两个图片。风格如图 7-4 中编号 2 的效果所示。样式表设计如下：

```
<style type="text/css" media="screen, projection">
@media (min-width:600px) and (max-width:900px) {
nav {
```

```
    float:left;
    width:35%;
}
section {
    float:left;
    width: 65%;
    padding:20px 0;
}
section li {
    float:left;
    margin:10px 2px;
    width:48%;
}
section li span {
    display:block;
    text-align:center;
    font-size:12px;
}
}
</style>
```

6．设计屏幕宽度小于600px且大于400px的样式表

屏幕宽度小于 600px 且大于 400px 时，整体页面结构是从上到下排列的：logo 在最上部，接下来是导航栏，再下面是图片列表，最下面是底部区域。其中，图片列表每行显示两个图片。风格如图 7-4 中编号 3 的效果所示。样式表设计如下：

```
<style type="text/css" media="screen, projection">
@media (min-width:400px) and (max-width: 600px) {
nav li {
    float:left;
    width:32%;
    margin-left:1%;
}
section {
    clear:both;
    padding:20px 0;
}
section li {
    float:left;
    margin:10px 2px;
    width:48%;
}
section li span {
    display:block;
    text-align:center;
    font-size:12px;
}
}
</style>
```

7．设计屏幕宽度小于400px的样式表

屏幕宽度小于 400px 时，整体页面结构也是从上到下排列的，与前面不同的是，图片列表每行只显示一个图片。风格如图 7-4 中编号 4 的效果所示。样式表设计如下：

```
<style type="text/css" media="screen, projection">
@media (man-width:400px) {
nav li {
```

```
    float:left;
    width:32%;
    margin-left:1%;
}
section {
    clear:both;
    padding:20px 0;
}
section li {
    margin:10px 2px;
}
section li span {
    display:block;
    text-align:center;
    font-size:12px;
}
}
</style>
```

至此，页面设计完毕。当改变浏览器窗口的宽度时，页面会根据媒体查询的结果应用不同的样式表，页面布局会发生改变。

7.3　小　　结

本章主要讲解媒体查询及其使用方法。重点讲解了@media 规则及其支持的设备范围，以及如何使用媒体查询。本章的难点在于对多种设备的理解，因为读者可能对很多设备都比较陌生；另外使用媒体查询的语法规则比较复杂，也难以理解，只有通过示例演示，才能更好地理解其语法规则。

下一章将介绍 HTML 5 的绘图功能——Canvas 绘图。

7.4　习　　题

【习题 1】CSS 3 新增了 Media Queries（媒体查询）模块，该模块使用什么规则来区别媒体设备？

【习题 2】简要说明媒体查询的优势。

【习题 3】编写一个包含媒体查询的页面，当尺寸改变时，会呈现不同的背景颜色。

第 3 篇　基于 HTML 5 的 Web 应用开发实战

第 8 章　绘制图形如此简单

绘制图形有很多方法，可以借助 Flash 实现，也可以使用 SVG 和 VML 来绘图。本章将学习一种新的绘图方法——使用 Canvas 元素，它是基于 HTML 5 原生的绘图功能。使用 Canvas 元素，可以绘制图形，也可以实现动画。它方便了使用 JavaScript 脚本的前端开发人员，寥寥数行代码，就可以在 Canvas 元素中实现各种图像及动画。本章将介绍如何使用 Canvas 元素来绘制一些简单的图形。

8.1　Canvas 简介

HTML 5 的 Canvas 元素有一套绘图 API（即接口函数），自成体系。JavaScript 就是通过调用这些绘图 API 来实现绘制图形和动画功能的。

1．Canvas的历史

在 HTML 5 以前的标准中，都有一个缺陷，就是不能直接动态地在 HTML 页面中绘制图形。若要在页面中实现绘图，或者是非常复杂的页面实现（使用大量的 JavaScript 代码），要么借助第三方工具实现（如 Flash、SVG、VML 等）。这种做法无疑把问题复杂化了。在互联网应用不断的发展中，页面绘图使用得越来越多。未来的发展趋势，也需要 HTML 自己完成绘图功能，Canvas 元素应运而生。

Canvas 元素是为了客户端矢量图形而设计的。它自己没有行为，但却把一个绘图 API 展现给客户端 JavaScript，以使脚本能够把想绘制的东西都绘制到一块画布上。Canvas 的概念最初是由苹果公司提出的，并在 Safari1.3Web 浏览器中首次引入。随后 Firefox 1.5 和 Opera 9 两款浏览器都开始支持 Canvas 绘图。目前 IE 9 也已经支持这项功能。Canvas 的标准化由一个 Web 浏览器厂商的非正式协会在推进，目前（Canvas）已经成为 HTML 5 草案中一个正式的标签。

2．Canvas和SVG及VML之间的差异

Canvas 有一个基于 JavaScript 的绘图 API，而 SVG 和 VML 使用一个 XML 文档来描述绘图。Canvas 与 SVG 和 VML 的实现方式不同，但在实现上可以相互模拟。Canvas 有自己的优势，由于不存储文档对象，性能较好。但若要移除画布里的图形元素，往往需要擦掉绘图重新绘制它。

8.2　Canvas 基本知识

在网页上使用 Canvas 元素，像使用其他 HTML 标签一样简单，然后利用 JavaScript 脚本调用绘图 API，绘制各种图形。Canvas 拥有多种绘制路径、矩形、圆形、字符及添加图像的方法，还能实现动画。下面，还是让我们从最简单的开始吧。

8.2.1　构建 Canvas 元素

Canvas 元素是以标签的形式应用到 HTML 页面里的。在 HTML 页面中放入如下代码即可。

```
<canvas></canvas>
```

不过 Canvas 元素毕竟是个新东西，很多旧的浏览器都不支持。为了增加用户体验，可以提供替代文字，放在 Canvas 标签中。例如：

```
<canvas>你的浏览器不支持该功能！</canvas>
```

当浏览器不支持 Canvas 元素时，标签里的文字就会显示出来。跟其他 HTML 标签一样，Canvas 标签有一些共同的属性：

```
<canvas id="canva" width="200" height="200">你的浏览器不支持该功能！</canvas>
```

其中，id 属性决定了 Canvas 标签的唯一性，方便查找。width 和 height 属性分别决定了 Canvas 的宽和高，其数值代表 Canvas 标签内包含了多少像素。

Canvas 标签可以像其他标签一样应用 CSS 样式表。如果在头部的样式表中添加如下样式：

```
canvas{
    border:1px solid #ccc;
}
```

那么该页面中的 Canvas 标签将有一个边框。

【示例 8-1】 在页面中构建 Canvas 元素。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>Canvas 标签使用</title>
<style type="text/css">
canvas{
    border:1px solid #ccc; /* 设置 Canvas 标签的边框样式*/
}
</style>
</head>
<body>
```

```
<canvas id="canva" width="200" height="200">你的浏览器不支持该功能！</canvas>
</body>
</html>
```

运行结果如图 8-1 和图 8-2 所示。



图 8-1　Canvas 在浏览器中显示的效果

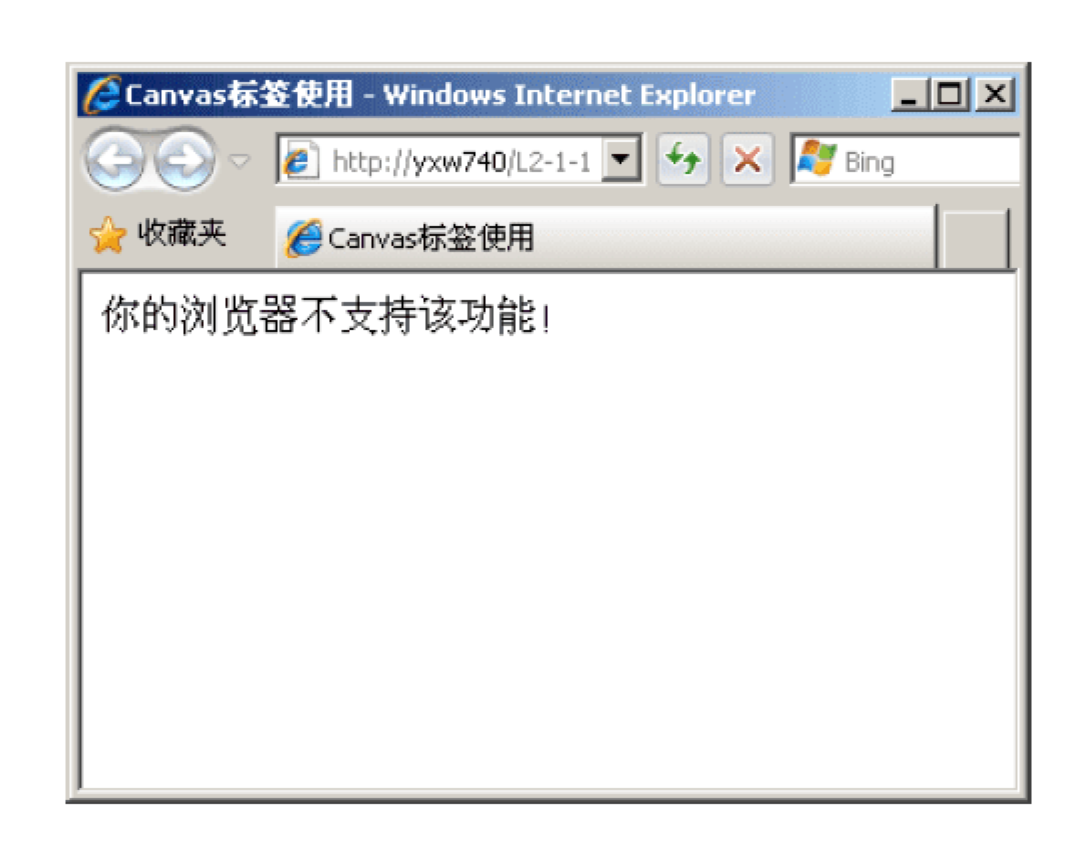


图 8-2　浏览器不支持 Canvas 时显示的效果

提示：也可以使用 CSS 样式来控制 Canvas 的宽和高，但 Canvas 内部的像素点还是根据 Canvas 自身的 width 和 height 属性确定，默认时是宽 300 像素，高 150 像素。用 CSS 设置 Canvas 尺寸，只能体现 Canvas 占用的页面空间，但是 Canvas 内部的绘图像素还是由 width 和 height 属性来决定的。这样会导致整个 Canvas 内部的图像变形。

8.2.2　使用 JavaScript 实现绘图的流程

Canvas 元素本身是没有绘图能力的。所有的绘制工作必须在 JavaScript 内部完成。前面讲过，Canvas 元素提供了一套绘图 API。在开始绘图之前，先要获取 Canvas 元素的对象，再获取一个绘图上下文，接下来就可以使用绘图 API 中丰富的功能了。

1．获取Canvas对象

在绘图之前，首先需要从页面中获取 Canvas 对象。通常使用 document 对象的 getElementById()方法获取。例如，以下代码获取 id 为“canvas”的 Canvas 对象。

```
var canvas=document.getElementById("canvas");
```

开发者还可以使用通过标签名称来获取对象的 getElementsByTagName 方法。

2．创建二维的绘图上下文对象

Canvas 对象包含了不同类型的绘图 API，还需要使用 getContext()方法来获取接下来要使用的绘图上下文对象。

```
var context=canvas.getContext("2d");
```

getContext 对象是内建的 HTML 5 对象，拥有多种绘制路径、矩形、圆形、字符及添加图像的方法。参数为“2d”，说明接下来将绘制的是一个二维图形。

3. 在Canvas上绘制文字

设置绘制文字的字体样式、颜色和对齐方式，然后将文字“囧”绘制在中央位置。

```
//设置字体样式、颜色及对齐方式
context.font="98px 黑体";
context.fillStyle="#036";
context.textAlign="center";
//绘制文字
context.fillText("囧",100,120,200);
```

font 属性设置了字体样式。fillStyle 属性设置了字体颜色。textAlign 属性设置了对其方式。fillText()方法用填充的方式在 Canvas 上绘制了文字。

【示例 8-2】 下面的代码首先在页面中绘制一个边框为 1 的 Canvas 对象，并使用 JavaScript 代码在其中绘制一个大大的“囧”字。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Canvas 标签使用</title>
<style type="text/css">
canvas{
    border:1px solid #ccc;
}
</style>
<script type="text/javascript">
function DrawText(){
    var canvas=document.getElementById("canvas");
    var context=canvas.getContext("2d");
    //设置字体样式、颜色及对齐方式
    context.font="98px 黑体";
    context.fillStyle="#036";
    context.textAlign="center";
    //执行绘制
    context.fillText("囧",100,120,200);
}
window.addEventListener("load", DrawText,true);
</script>
</head>
<body style="text-align:center">
<canvas id="canvas" width="200" height="200">你的浏览器不支持该功能！</canvas>
</body>
</html>
```

运行结果如图 8-3 所示。



图 8-3　在 Canvas 中绘制的文字“囧”

绘制出来了，这是不同于以往的实现方法，这是一个很好的开始。当然，这种绘制方式完全借助了绘图 API。

8.3　使用 Canvas 绘图

本节将深入学习使用绘图 API。在接下来的小节中将逐个演绎 Canvas 的各种绘图功能。

8.3.1　绘制矩形

矩形属于一种特殊而又普遍使用的一种图形。矩形的宽和高，就确定了图形的样子。再给予一个绘制起始坐标，就可以确定其位置。这样整个矩形就确定下来了，本小节详细讲解矩形的绘制。绘图 API 为绘制矩形提供了两个专用的方法：strokeRect()和 fillRect()，可分别用于绘制矩形边框和填充矩形区域。在绘制之前，往往需要先设置样式，然后才能进行绘制。

1．设置样式

关于矩形可以设置的属性有：边框颜色、边框宽度、填充的颜色等。绘图 API 提供了几个属性可以设置这些样式，属性说明如表 8.1 所示。

表 8.1　常用属性

属　　性	取　　值	说　　明
strokeStyle	符合 CSS 规范的颜色值及对象	设置线条的颜色
lineWidth	数字	设置线条宽度，默认宽度为 1，单位是像素
fillStyle	符合 CSS 规范的颜色值	设置区域或文字的填充颜色

其中，strokeStyle 可设置矩形边框的颜色，lineWidth 可设置边框宽度，fillStyle 可设置填充颜色。

2．绘制矩形边框

绘制矩形边框需要使用 strokeRect 方法。其语法如下：

```
strokeRect(x,y,width,height);
```

其中，width 表示矩形的宽度，height 表示矩形的高度，x 和 y 分别是矩形起点的横坐标和纵坐标。例如，以下代码以(50,50)为起点绘制一个宽度为 150，高度为 100 的矩形。

```
context.strokeRect(50,50,150,100);
```

这里仅仅绘制了矩形的边框，且边框的颜色和宽度由属性 strokeStyle 和 lineWidth 来指定。

3. 填充矩形区域

填充矩形区域需要使用 fillRect()方法。其语法如下：

```
fillRect(x,y,width,height);
```

该方法的参数和 strokeRect()方法的参数是一样的，用以确定矩形的位置及大小。例如，以下代码以(50,50)为起点绘制一个宽度为 150，高度为 100 的矩形。

```
context.fillRect(50,50,150,100);
```

这里填充了一个矩形区域，且填充的颜色由属性 fillStyle 来指定。

【示例 8-3】 绘制一个黄色的矩形，边框为黑色。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>绘制一个黄色的矩形，边框为黑色</title>
<style type="text/css">
canvas {
    border:1px solid #000;
}
</style>
<script type="text/javascript">
function DrawRect(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //绘制矩形边框
    context.strokeStyle="#000";               //设置边框颜色
    context.lineWidth=1;                      //指定边框宽度
    context.strokeRect(50,50,150,100);        //绘制矩形边框
    //填充矩形形状
    context.fillStyle="#f90";                 //设置填充颜色
    context.fillRect(50,50,150,100);          //填充矩形区域
}
window.addEventListener("load",DrawRect,true);
</script>
</head>
<body style="overflow:hidden">
<canvas id="canvas" width="400" height="300">你的浏览器不支持该功能！</canvas>
</body>
</html>
```

运行结果如图 8-4 所示。



图 8-4　在 Canvas 中绘制矩形边框及填充效果

代码分析：在绘制矩形的过程中，strokeRect()方法是用来绘制边框的，fillRect()方法是用来填充区域的，而且各自都指定了矩形区域，是两个不同的绘图方法，两个绘制过程。在这两次绘制过程中，设置了同样位置和同样大小的矩形，看起来像是在矩形框里填充了相应的颜色。

4．特殊的矩形绘制方式

除了本节介绍的两个与矩形有关的方法（绘制矩形边框和填充矩形区域）外，还有一个方法 clearRect。执行该方法，将会擦除指定的矩形区域，使其变为透明。使用方法如下：

```
context.clearRect (x,y,width,height);
```

该方法中的 4 个参数请参考 strokeRect()方法和 fillRect()方法，代表的意义一样。加入代码如下：

```
context.clearRect (60,60,100,50);
```

运行结果如图 8-5 所示。



图 8-5　在 Canvas 中的擦除效果

提示： 本节绘制矩形使用了两种绘图方式：绘制线条和填充区域。绘制线条，无论是绘制矩形还是其他图形，都是类似的用法，有共同的属性设置。填充区域，也有共同的属性设置，如属性 fillStyle，在其他填充方式中也需要用到这个属性。

8.3.2　使用路径

路径就是预先构建的图像轮廓。它由一个或多个直线段或曲线段组成，可以是开放的，也可以是闭合的。在很多绘图工具或方法中都会使用，如 Photoshop 中的路径。路径会在实际绘图前勾勒出图形的轮廓，这样就可以绘制复杂的图形。

在 Canvas 中，所有基本图形都是以路径为基础的，我们通常会调用 lineTo()、rect()、arc()等方法来设置一些路径。在最后使用 fill()或 stroke()方法进行绘制边框或填充区域时，都是参照这个路径来进行的。使用路径绘图基本上分如下三个步骤。

- 创建绘图路径。
- 设置绘图样式。
- 绘制图形。

与上一节绘制矩形的方法对比，这里多了创建绘图路径的步骤。绘图 API 为创建路径提供了很多宝贵的方法，下面进行详细讲解。

1. 创建绘图路径

创建绘图路径经常会用到两个方法 beginPath()和 closePath()，分别表示开始一个新的路径和关闭当前的路径。首先，使用 beginPath()方法创建一个新的路径。该路径是以一组子路径的形式储存的，它们共同构成一个图形。每次调用 beginPath()方法，都会产生一个新的子路径。语法如下所示：

```
context.beginPath();
```

接着就可以使用多种设置路径的方法。绘图 API 提供了丰富的路径方法，如表 8.2 所示。

表 8.2　常用的路径方法

方　　法	参　　数	说　　明
moveTo(x,y)	x,y 确定了起始坐标	绘图开始的坐标
lineTo(x,y)	x,y 确定了直线路径的目标坐标	绘制直线到目标坐标
arc(x, y, radius, startAngle,endAngle, counterclockwise)	x, y 描述弧的圆形的圆心的坐标；radius 描述弧的圆形的半径；startAngle 圆弧的开始点的角度；endAngle 圆弧的结束点的角度；counterclockwise 逆时针方向 true，顺时针方向 false	使用一个中心点和半径，为一个画布的当前路径添加一条弧线。圆形为弧形的特例
rect(x,y,width,height)	x, y 描述矩形起点坐标；width,height 描述矩形的宽和高	矩形路径方法

最后，使用 closePath()方法关闭当前路径。语法如下：

```
context.closePath();
```

它会尝试用直线连接当前端点与起始端点来闭合当前路径，但是如果当前路径已经闭

合或者只有一个点，则什么都不做。

2. 设置绘图样式

设置绘图样式包括边框样式和填充样式。其形式如下：

（1）使用 strokeStyle 属性设置矩形边框的颜色，设置边框颜色为黑色：

```
context.strokeStyle="#000";
```

（2）使用 lineWidth 属性设置边框宽度，设置边框宽度为 3 像素：

```
context.lineWidth=3;
```

（3）使用 fillStyle 属性设置填充颜色，设置填充颜色为橘黄色：

```
context.fillStyle="#f90";
```

3. 绘制图形

路径和样式都设置好了，最后就是调用方法 stroke()绘制边框，或调用方法 fill()填充区域。

```
context.stroke();                              //绘制边框
context.fill();                                //填充区域
```

这时，图形才实际地绘制到 canvas 上去。

【示例 8-4】 绘制一个圆形和矩形叠加的图形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");

    //创建绘图路径
    context.beginPath();                       //创建一个新的路径
    context.arc(150,100,50,0,Math.PI*2,true);  //圆形路径
    context.rect(50,50,100,100);               //矩形路径
    context.closePath();

    //设置样式
    context.strokeStyle="#000";                //设置边框颜色
    context.lineWidth=3;                       //设置边框宽度
    context.fillStyle="#f90";                  //设置填充颜色

    //填充矩形形状
    context.stroke();                          //绘制边框
    context.fill();                            //填充区域
}
```

提示：在示例 8-4 中，仅列出了核心的脚本代码。因为页面的其他部分是不变的，只有绘图过程，因绘制不同的图形才会改变。为节省篇幅，只保留 JavaScript 代码。

运行结果如图 8-6 所示。

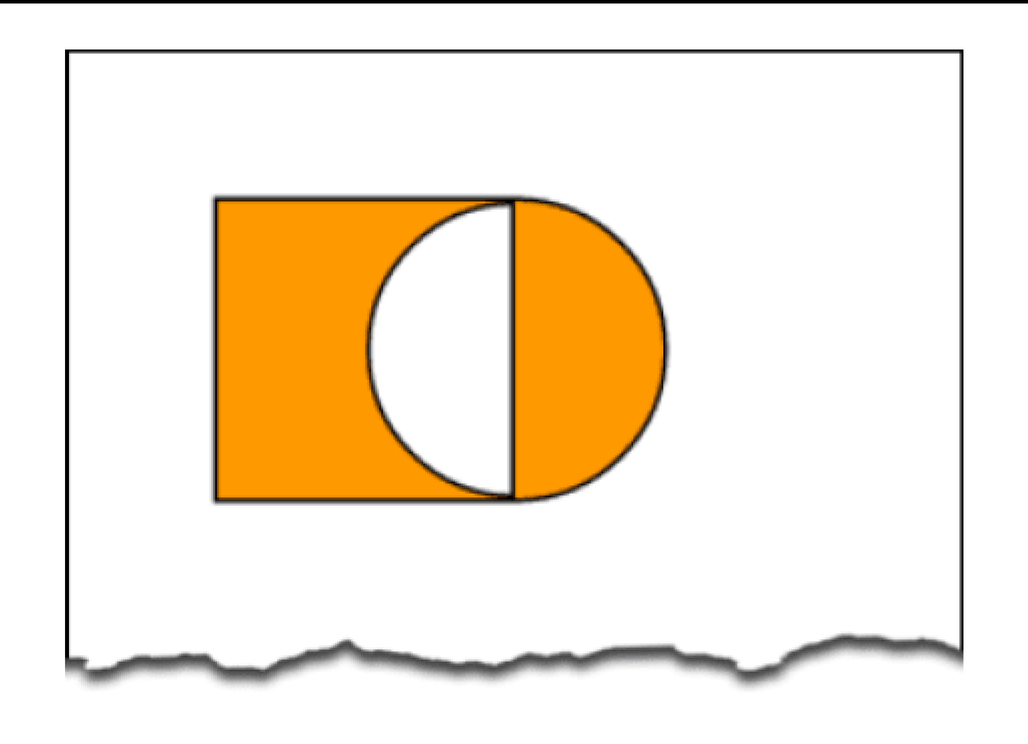


图 8-6　圆形和矩形叠加效果

代码分析：在创建路径的过程中，分别使用了 arc 方法和 rect 方法，创建了一个圆形和一个矩形，并且其中有重叠的部分为空白。最后调用了方法 stroke()和方法 fill()，完成了边框的绘制及区域填充。

现在理解一下重叠的空白部分。修改一下圆形路径的参数，如下：

```
context.arc(100,100,30,0,Math.PI*2,true);        //圆形路径
```

运行结果如图 8-7 所示。

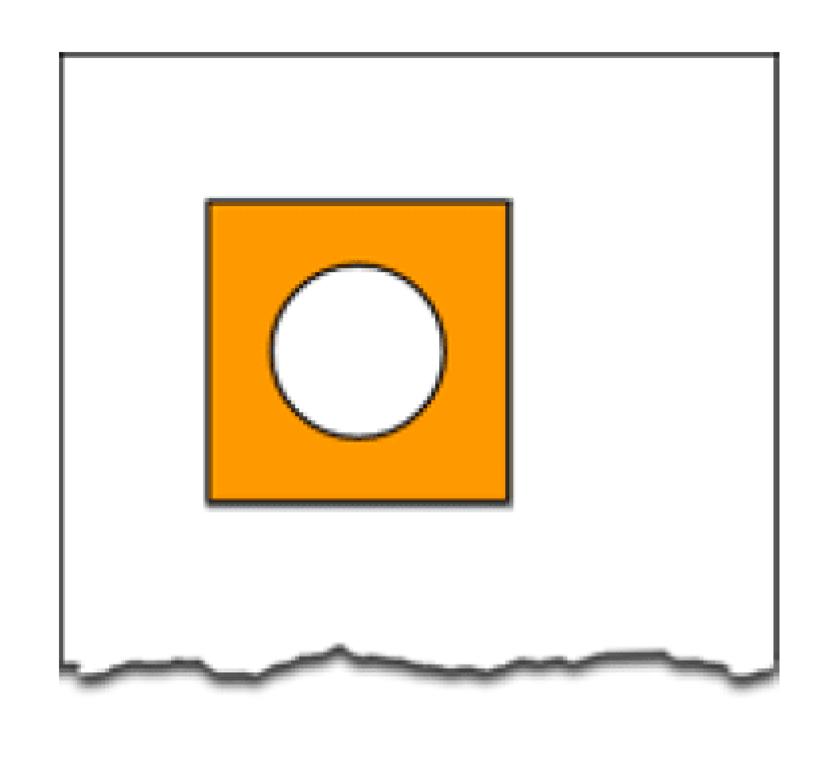


图 8-7　圆形和矩形叠加效果

从图 8-7 中更容易理解，填充的部分为矩形和圆形之间的区域，而这部分区域正是路径所确定的区域。即可以这样理解，当两个图形路径重叠时，重叠区域则被排除在路径确定的区域之外。

4．深入理解路径特性

在创建子路径的三个步骤中，展示的是一个标准的子路径创建过程。如果绘制复杂的图形，标准化的方法有利于代码的规范和管理。

（1）理解一下 beginPath()方法的作用。

使用 beginPath()方法，可以新建一个子路径，接下来的绘制，都是针对该子路径进行的。如果不使用该方法，那么设置的路径和前面的路径设置默认为同一个路径设置。在接下来的绘制中，前面设置的路径会被重复绘制。

为了体现重复绘制的区别，接下来，会使用半透明的颜色样式设置。下面就使用路径绘图的方法分别绘制两个圆形，填充为半透明的颜色，代码如示例 8-5 所示。

【示例 8-5】 绘制两个半透明的圆形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");

    //绘制第一个圆形
    context.beginPath();
    context.arc(150,100,50,0,Math.PI*2,true);
    context.fillStyle="rgba(255,135,0,0.4)";
    context.fill();

    //绘制第二个圆形
    context.beginPath();
    context.arc(170,120,50,0,Math.PI*2,true);
    context.fillStyle="rgba(255,135,0,0.4)";
    context.fill();
}
```

运行结果如图 8-8 所示。如果去掉示例 8-5 中的 beginPath()方法，运行结果如图 8-9 所示。

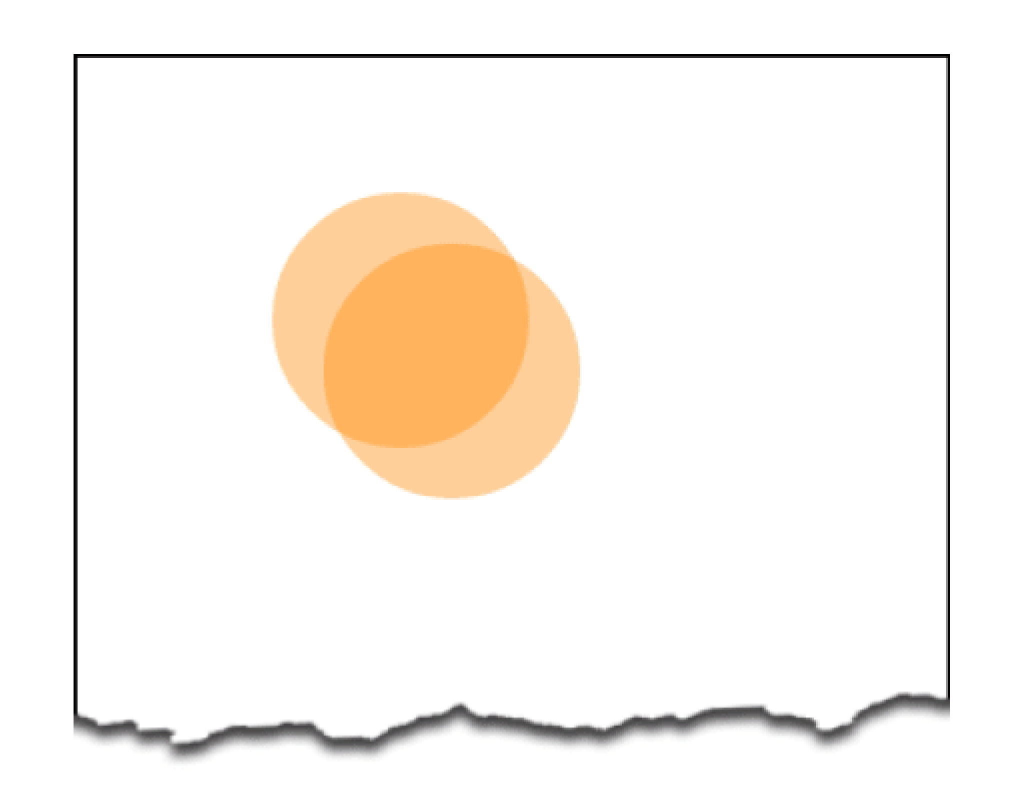


图 8-8　绘制的两个半透明的圆形

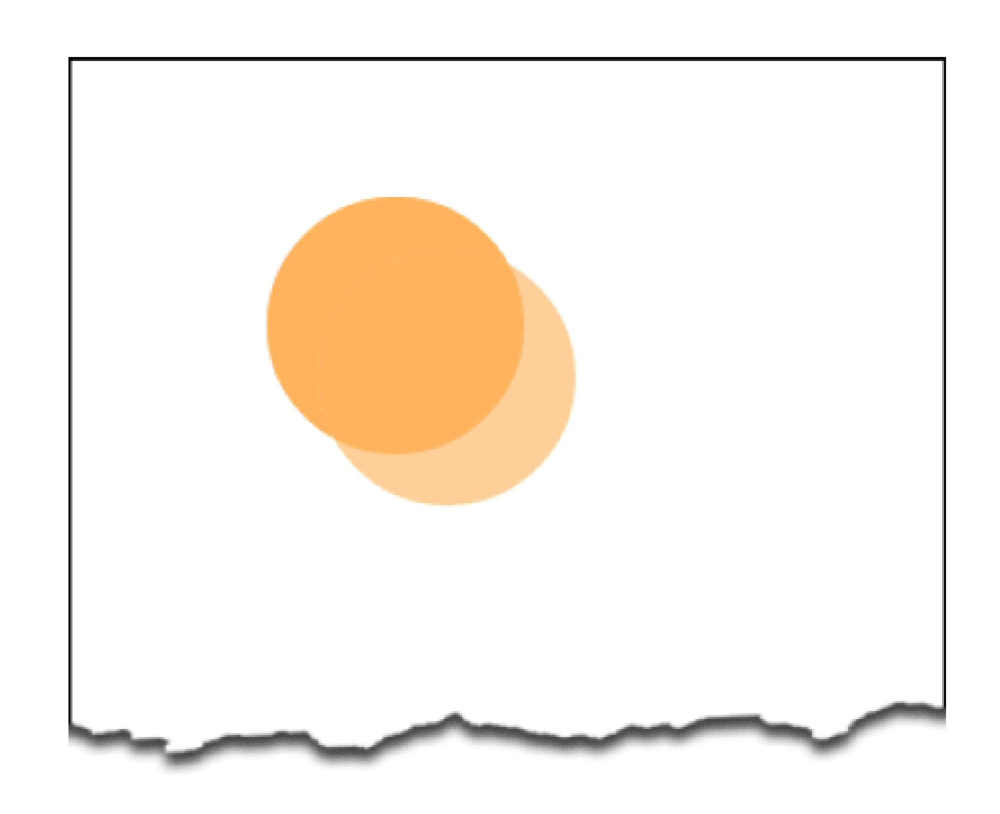


图 8-9　绘制的两个半透明的圆形

代码分析：在图 8-8 中，通过两个子路径，绘制了两个圆形，两个圆形重叠部分的颜色叠加显示。在图 8-9 中，去除了 beginPath()方法，绘制图形时就不再创建子路径。第一个圆形，在执行过程中，将被填充两次。所以，按照规范的绘制方法，可以防止绘图混乱，以至于绘制出意想不到的图形。不过某种情况下，也许可以利用这一特性。

（2）理解一下 closePath()方法的作用。

closePath()方法是用来闭合路径的。如果前面设置的路径是开放的，closePath()方法会自动用直线连接终点和起点。同样，使用路径的方法绘制两条直线。使用 moveTo(x,y)方法设置当前坐标位置；使用 lineTo(x, y)方法为当前子路径添加一条直线。

【示例 8-6】 绘制两条直线。

```
function DrawLine(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //创建绘制过程
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(120,120);
```

```
    context.lineTo(120,60);
    context.closePath();
    context.strokeStyle ="#000";
    //执行绘制
    context.stroke();
}
```

运行结果如图 8-10 所示。如果去掉示例 8-6 中的 closePath()方法，运行结果如图 8-11 所示。

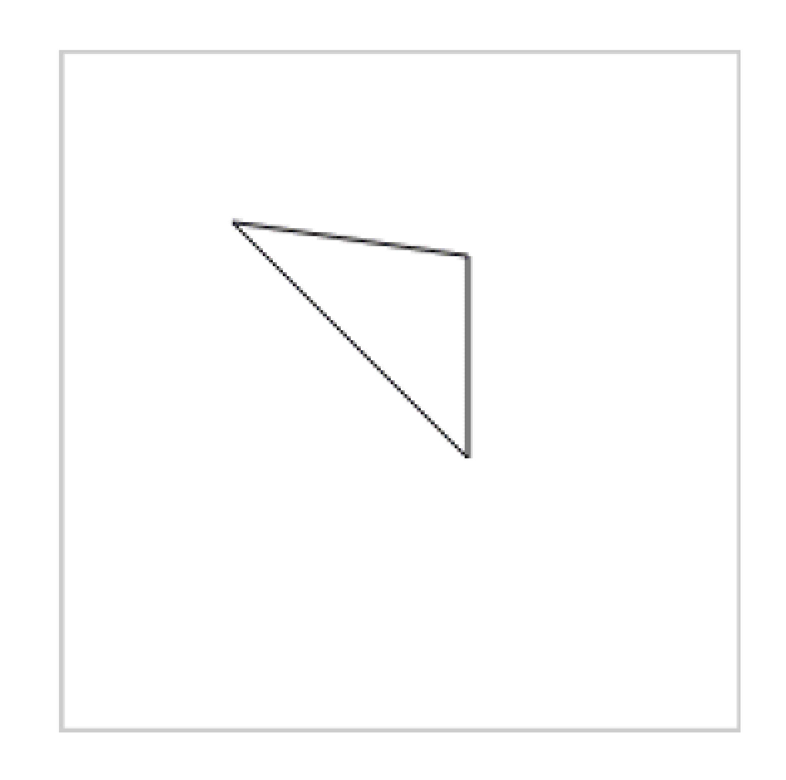


图 8-10　绘制两条直线（闭合路径）

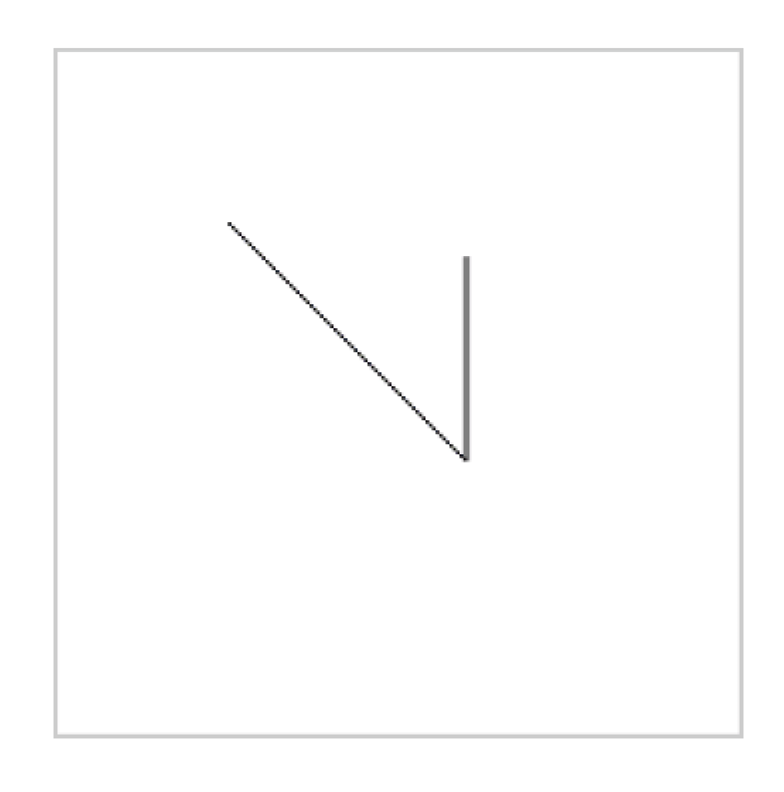


图 8-11　绘制两条直线（开放路径）

代码说明：方法 closePath()，会自动用一条直线连接路径的起始点，以完成路径的闭合。如果需要绘制一个开放的路径，就不要使用它了。

提示：closePath()方法习惯性地放在路径设置的最后一步，切勿认为是路径设置的结束。因为在此之后，还可以继续设置路径。

8.3.3　图形组合

通常，会把一个图像绘制在另一个图形之上，称之为图形组合。默认的情况是上面的图形覆盖了下面的图形，这是由于图形组合默认地设置了“source-over”。

在 Canvas 中，可通过属性 globalCompositeOperation 来设置如何在画布上组合颜色，总共有 12 种组合类型。语法如下：

```
globalCompositeOperation= [value];
```

参数说明：参数 value 的合法值有 12 个，决定了 12 种图形组合类型，默认值是“source-over”。12 种组合类型如表 8.3 所示。

表 8.3　组合类型值的含义

值	含　义
copy	只绘制新图形，删除其他所有内容
darker	在图形重叠的地方，颜色由两个颜色值相减后决定
destination-atop	已有的内容只在它和新的图形重叠的地方保留。新图形绘制于内容之后
destination-in	在新图形以及已有画布重叠的地方，已有内容都保留。所有其他内容成为透明的
destination-out	在已有内容和新图形不重叠的地方，已有内容保留。所有其他内容成为透明的

续表

值	含　　义
destination-over	新图形绘制于已有内容的后面
lighter	在图形重叠的地方，颜色由两种颜色值的加值来决定
source-atop	只有在新图形和已有内容重叠的地方，才绘制新图形
source-in	只有在新图形和已有内容重叠的地方，新图形才绘制。所有其他内容成为透明的
source-out	只有在和已有图形不重叠的地方，才绘制新图形
source-over	新图形绘制于已有图形的顶部。这是默认的行为
xor	在重叠和正常绘制的其他地方，图形都成为透明的

【示例 8-7】 多样化的图形组合。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //source-over
    context.globalCompositeOperation = "source-over";
    RectArc(context);
    //lighter
    context.globalCompositeOperation = "lighter";
    context.translate(90,0);
    RectArc(context);
    //xor
    context.globalCompositeOperation = "xor";
    context.translate(-90,90);
    RectArc(context);
    //destination-over
    context.globalCompositeOperation = "destination-over";
    context.translate(90,0);
    RectArc(context);
}
//绘制组合图形
function RectArc(context){
    context.beginPath();
    context.rect(10,10,50,50);
    context.fillStyle = "#F90";
    context.fill();
    context.beginPath();
    context.arc(60,60,30,0,Math.PI*2,true);
    context.fillStyle = "#0f0";
    context.fill();
}
```

运行结果如图 8-12 所示。

代码分析：函数 RectArc(context)是用来绘制组合图形的，使用方法 translate()移动不同的位置，连续绘制了 4 种组合图形：source-over、lighter、xor、destination-over，由图 8-12 可见组合效果。关于方法 translate()，将在后面的章节中讲解。

提示： 关于这 12 种图形组合方式，应参照表 8.3 中的描述。在示例 8-7 中列出的 4 种组合方式，在各个浏览器中的效果基本一致，其他图形组合方式，在各个浏览器中都有不同程度的偏差或不一致。虽然这是一个很好的功能，但在实际开发中应考虑不同浏览器的兼容性。

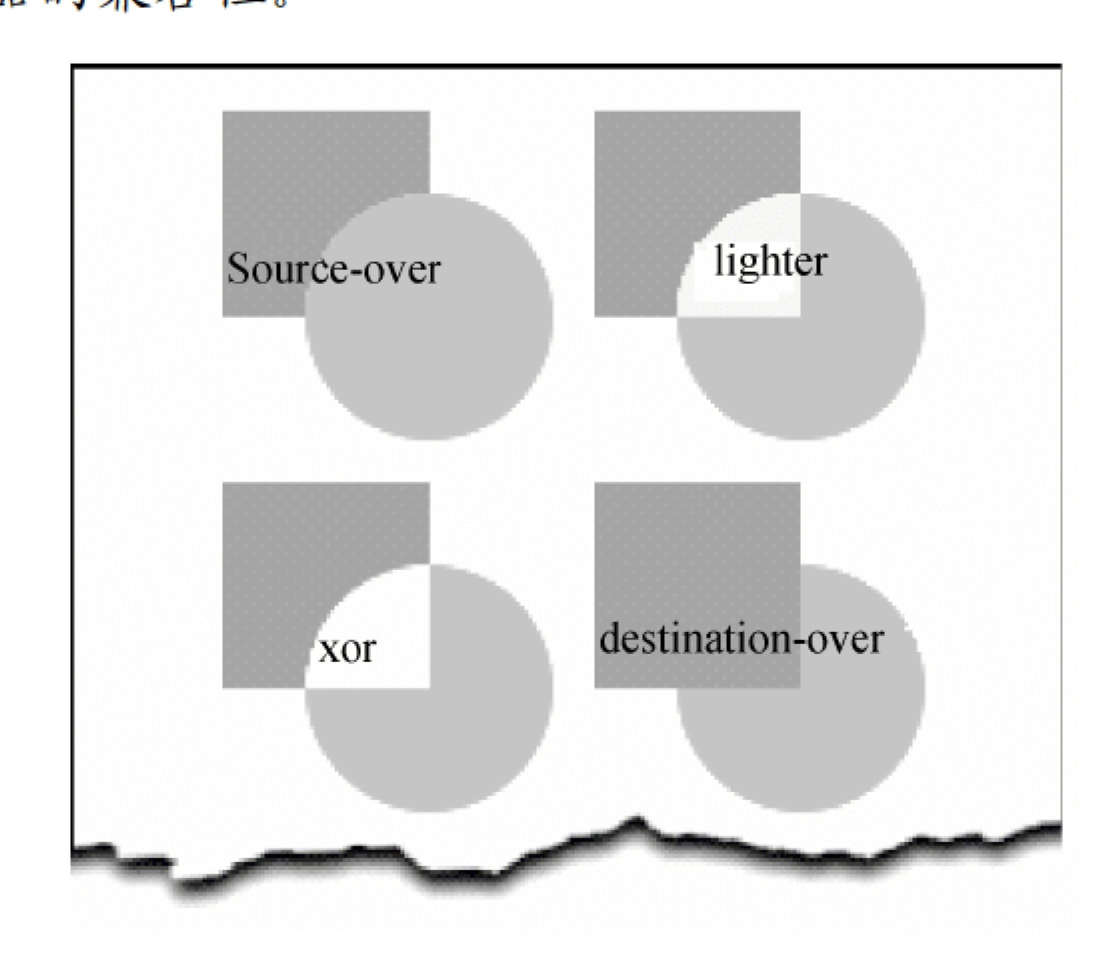


图 8-12　圆形和矩形叠加效果

8.3.4　绘制曲线

在实际的绘图中，绘制曲线是常用的一种绘图形式。我们在设置路径的时候，需要使用一些曲线方法来勾勒出曲线路径，以完成曲线的绘制。在 Canvas 中，绘图 API 提供了多种曲线绘制方法。主要的曲线绘制方法有 arc()、arcTo()、quadraticCurveTo()、bezierCurveTo()等。

1．使用中心点和半径绘制弧线——arc()方法

在上一节中，已经在应用 arc()方法绘制圆形。该方法是使用中心点和半径，为一个画布的当前路径添加一条弧线。语法如下：

```
arc(x, y, radius, startAngle,endAngle, counterclockwise);
```

参数说明：*x* 和 *y* 描述弧的圆形的圆心坐标。radius 描述弧的圆形的半径。startAngle 是圆弧的开始点的角度。endAngle 是圆弧的结束点的角度。counterclockwise 逆时针方向为 true，顺时针方向为 false。

如图 8-13 所示，圆心由参数 *x* 和 *y* 来确定，半径由参数 radius 确定，圆弧的开始点的角度 StartAngle 和结束点的角度 EndAngle 如图 8-13 中的箭头标注所示，体现的是一个逆时针方向的绘制。其中，沿着 *x* 轴正半轴的三点钟方向的角度为 0。

【示例 8-8】 使用 arc()方法绘制一条弧线。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //先绘制一个灰色的圆形
    context.beginPath();
```

```
    context.arc(150,100,50,0,Math.PI*2,true);
                                        //绘制一个圆心为(150,100)，半径为 50 的圆形
    context.fillStyle="rgba(0,0,0,0.1)";        //设置填充为黑色，透明度为 0.1
    context.fill();                             //填充 arc()方法确定的区域
    //再绘制一条圆弧，宽 5 像素，线条颜色为橘黄色
    context.beginPath();
    context.arc(150,100,50,0,(-Math.PI*2/3),true);
                                        //绘制一个圆心为(150,100)，半径为 50 的圆弧
    context.strokeStyle="rgba(255,135,0,1)";    //设置边框颜色为橘黄色
    context.lineWidth=5;                        //设置边框宽度为 5 像素
    context.stroke();                           //绘制 arc()方法确定的区域边框
}
```

运行结果如图 8-14 所示。

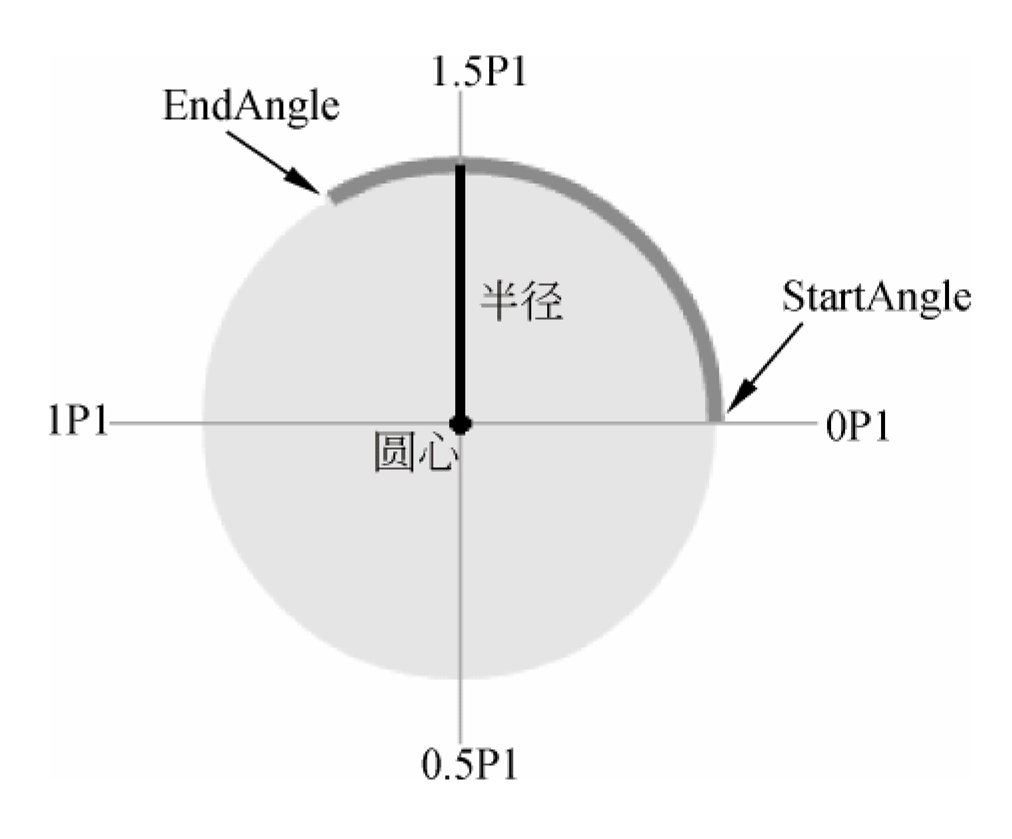




图 8-13　arc()方法绘制弧线原理解析图

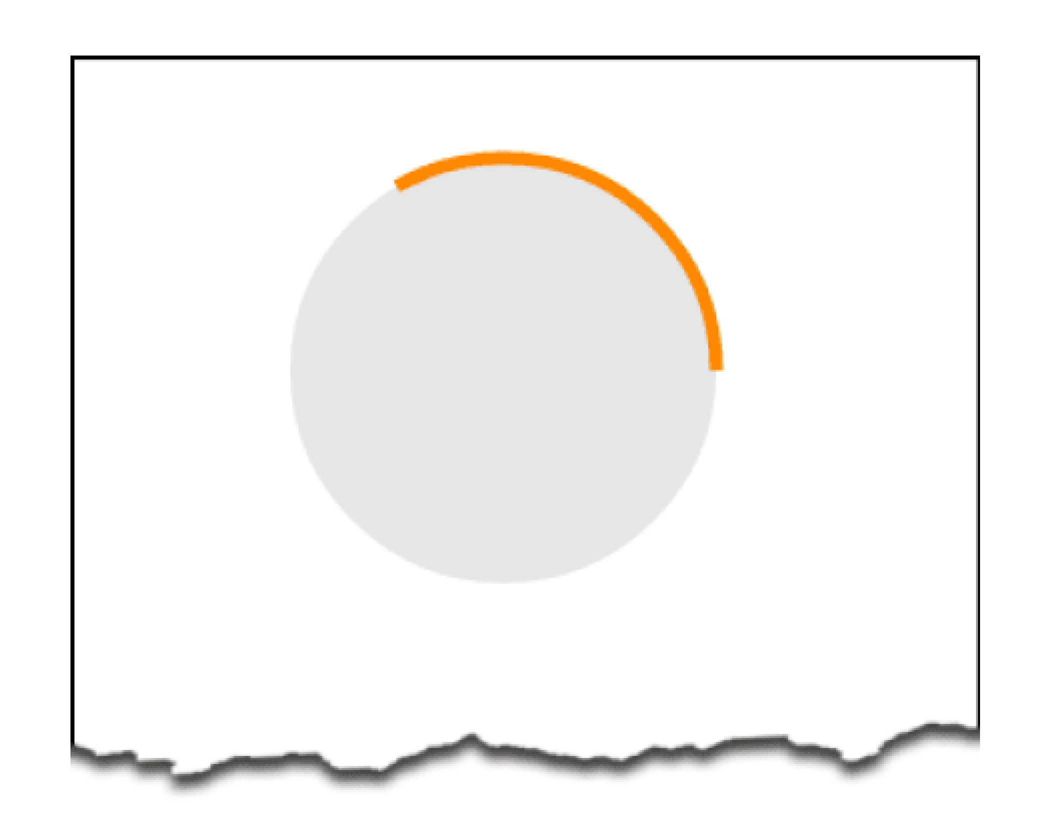


图 8-14　使用 arc()方法绘制一条橘黄色的弧线

代码分析：为了更好地说明问题，同时绘制一个灰色的圆形，圆形的圆心坐标和半径与弧线相同，弧线的线条宽 5 像素，线条颜色为橘黄色。在绘制弧线的时候，仅用 arc()方法就完成路径的设置，与其他路径的绘制一样，需要先设置填充样式或边框样式，最后执行填充或绘制。绘制弧线，不依赖于绘制起点，即不需要使用 moveTo()方法来确定绘制起点。

2．使用辅助线绘制弧线——arcTo()方法

arcTo()方法是使用切线的方法绘制弧线，使用两个目标点和一个半径，为当前的子路径添加一条弧线。与 arc()方法相比，同样是绘制弧线，绘制思路及侧重点不一样。语法如下：

```
arcTo(x1, y1, x2, y2, radius);
```

参数说明：*x*1, *y*1 描述了一个坐标点，用 *P*1 表示。*x*2, *y*2 描述了另一个坐标点，用 *P*2 表示。radius 描述弧的圆形的半径。如图 8-15 所示，有一个绘制的起点（即当前位置），通常会使用 moveTo()方法来指定。*P*1 点由参数 *x*1, *y*1 确定。*P*2 点由参数 *x*2, *y*2 确定。半径由参数 radius 确定。

添加给路径的圆弧是具有指定 radius 的圆的一部分。圆弧有一个点与起点到 *P*1 的线段相切（如图 8-15 所示的切点 1），还有一个点和从 *P*1 到 *P*2 的线段相切（如图 8-15 所示的切点 2）。这两个切点就是圆弧的起点和终点，圆弧绘制的方向就是连接这两个点的

最短圆弧的方向。

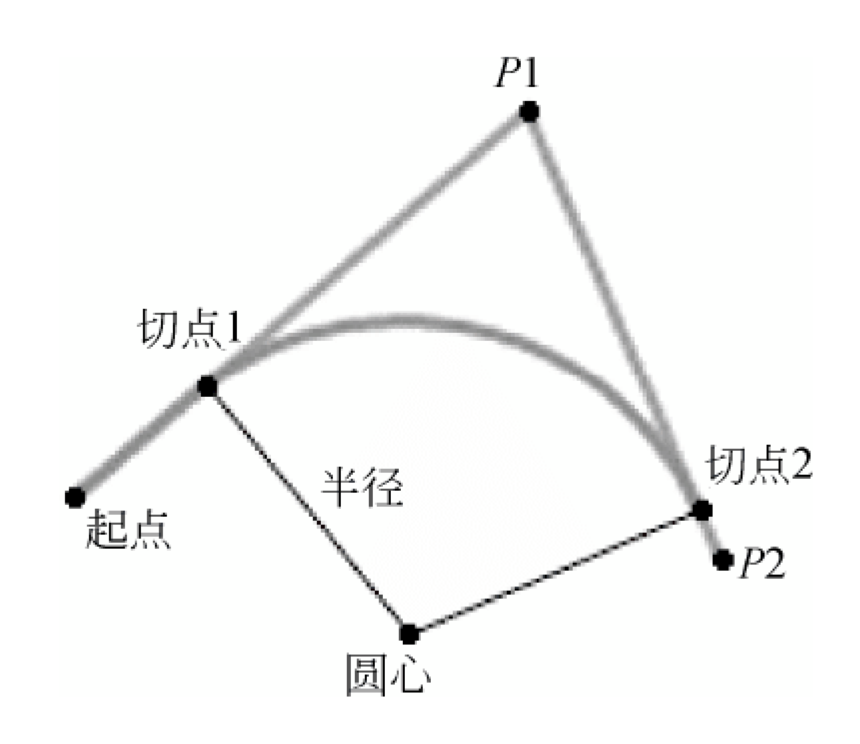


图 8-15　arcTo()方法绘制弧线原理解析图

从某种意义上讲，使用 arcTo()方法绘制圆弧借助了两条辅助线。下面使用 arcTo()方法来绘制一条弧线。

【示例 8-9】 使用 arcTo()方法绘制一条弧线。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //先绘制灰色的辅助线段，宽 2 像素
    context.beginPath();                         //添加第一个子路径
    context.moveTo(80,120);                      //确定当前位置，即绘图起始的位置
    context.lineTo(150,60);                      //到 P1 点的直线
    context.lineTo(180,130);                     //到 P2 点的直线
    context.strokeStyle="rgba(0,0,0,0.4)"; //线框颜色为黑色，透明度为 0.4
    context.lineWidth=2;                         //线框宽度为 2 像素
    context.stroke();
    //再绘制一条圆弧，宽 2 像素，线条颜色为橘黄色
    context.beginPath();                         //添加第二个子路径
    context.moveTo(80,120);                      //确定当前位置，与第一个子路径一致
    context.arcTo(150, 60, 180, 130, 50); //arcTo()方法确定弧线轮廓
    context.strokeStyle="rgba(255,135,0,1)";    //线框颜色为橘黄色
    context.stroke();
}
```

运行结果如图 8-16 所示。

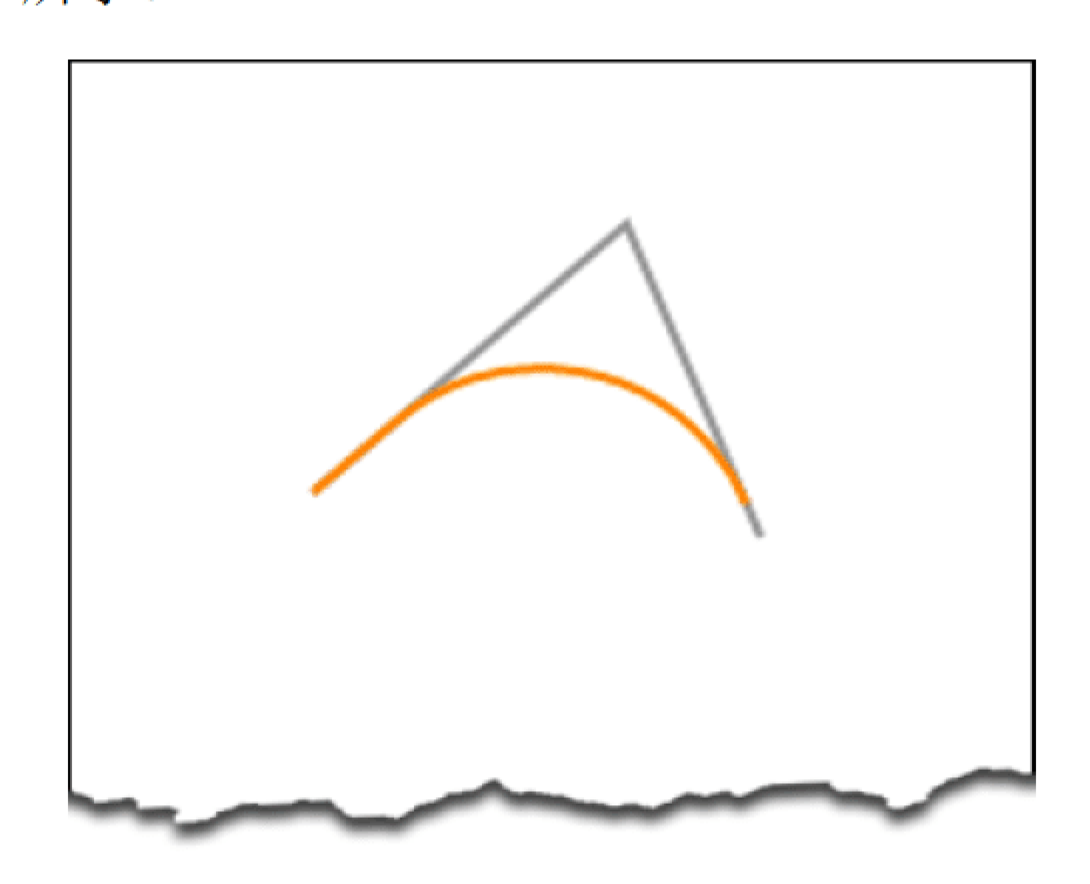


图 8-16　使用 arcTo()方法绘制一条橘黄色的弧线

代码分析：图 8-16 中的黄色部分为绘制的圆弧部分。由于该绘制弧线的方法是通过与辅助线段相切来完成的，所以将与弧线相切的辅助线段也绘制出来。在绘制弧线的时候，先通过 moveTo()方法确定了绘制起点。该起点会与圆弧的第一个切点连接起来。可以发现图 8-16 中的橘黄色线条有一段是直线。

3．绘制二次样条曲线——quadraticCurveTo()方法

二次样条曲线是曲线的一种，Canvas 绘图 API 专门提供了此曲线的绘制方法。quadraticCurveTo()方法为当前的子路径添加一条二次样条曲线。语法如下：

```
quadraticCurveTo(cpX, cpY,x,y);
```

参数说明：cpX、cpY 描述了控制点的坐标；*x*、*y* 描述了曲线的终点坐标。

如图 8-17 所示，起点即当前的位置，控制点由参数 cpX、cpY 确定，终点由参数 *x*、*y* 确定。这条橘黄色的曲线就是从起点连接到终点，而控制点可以控制起点和终点之间的曲线的形状。

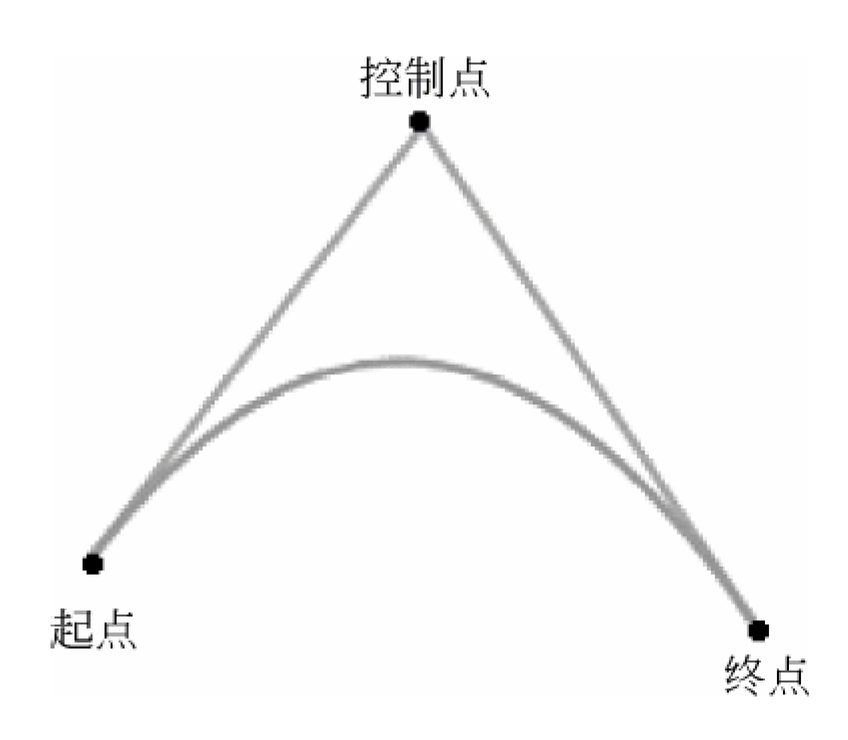


图 8-17　二次样条曲线原理解析图

下面使用 quadraticCurveTo()方法来绘制一条曲线。

【示例 8-10】 绘制二次样条曲线。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //先绘制灰色的辅助线段，宽 3 像素
    context.beginPath();                        //添加第一个子路径
    context.moveTo(100,180);                    //确定当前位置，即绘图起始的位置
    context.lineTo(200,50);                     //直线连接控制点
    context.lineTo(300,200);                    //直线连接终点
    context.strokeStyle="rgba(0,0,0,0.4)"; //线框颜色为黑色，透明度为 0.4
    context.lineWidth=3;
    context.stroke();
    //再绘制一条曲线，宽 3 像素，线条颜色为橘黄色
    context.beginPath();                        //添加第二个子路径
    context.moveTo(100, 180);                   //确定当前位置，与第一个子路径一致
    context.quadraticCurveTo(200, 50, 300, 200);   //确定曲线轮廓
    context.lineWidth = 3;
    context.strokeStyle="rgba(255,135,0,1)";       //线框颜色为橘黄色
    context.stroke();
}
```

运行结果如图 8-18 所示。

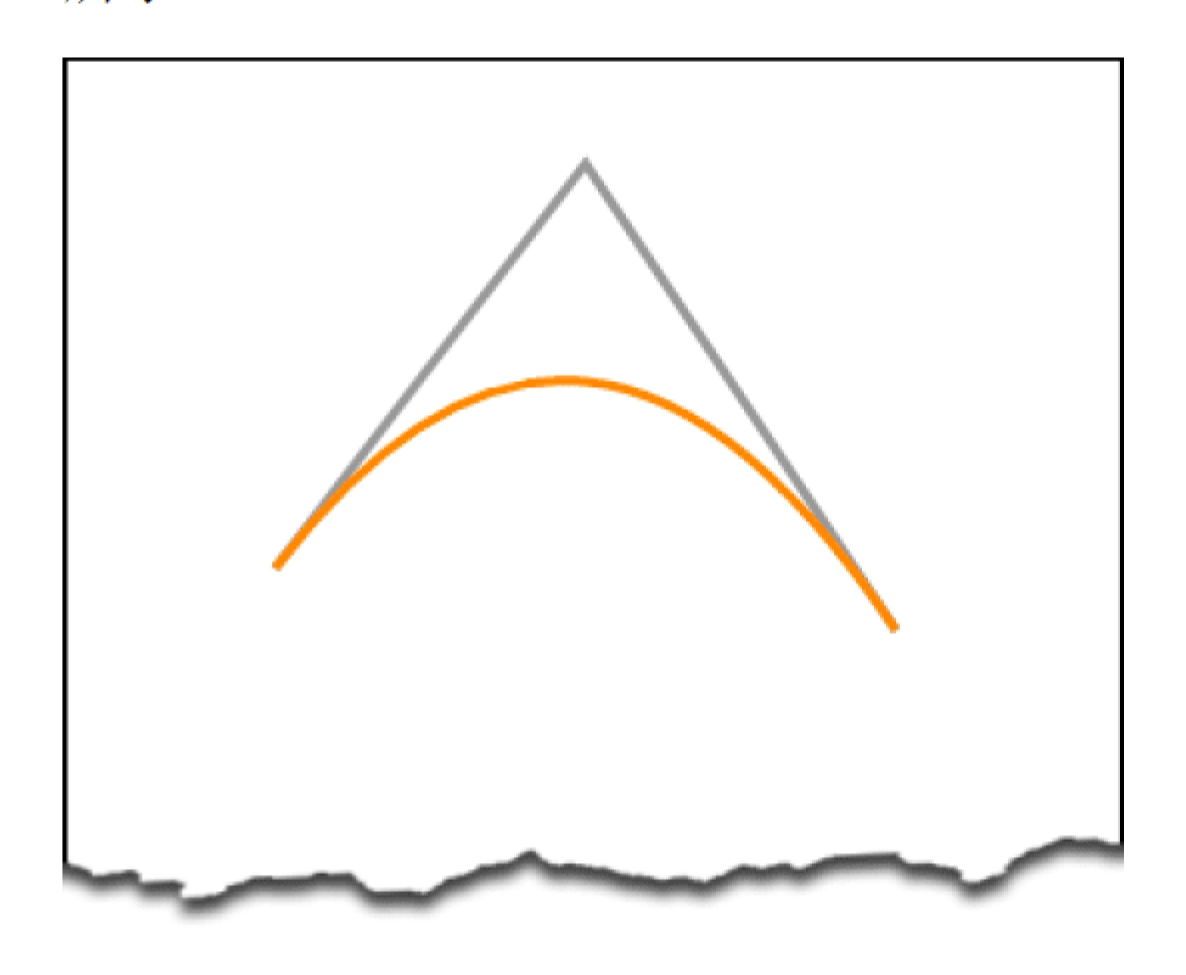


图 8-18　绘制的二次样条曲线

代码分析：由于该方法使用了两个坐标点，我们将这些点用直线连接起来，以辅助理解。在绘制曲线的时候，先通过 moveTo()方法确定绘制起点。图 8-18 中的黄色部分为绘制的曲线部分，连接了曲线的起点和终点，曲线的弯曲形状，由控制点控制。

提示：样条曲线是数学中的概念，已超出我们的研究范围，有兴趣的话可以去研究一下。

4．绘制贝塞尔曲线——bezierCurveTo()方法

贝塞尔曲线，又称贝兹曲线或贝济埃曲线，是应用于二维图形应用程序的数学曲线。Canvas 绘图 API 也提供了贝塞尔的绘制方法 bezierCurveTo()。与二次样条曲线相比，贝塞尔曲线使用了两个控制点，从而读者可以创建更复杂的曲线图形。语法如下：

```
bezierCurveTo(cp1X, cp1Y, cp2X, cp2Y,x,y);
```

参数说明：cp1X、cp1Y 描述了第一个控制点的坐标，cp2X、cp2Y 描述了第二个控制点的坐标，*x*、*y* 描述了曲线的终点坐标。

如图 8-19 所示，起点即当前的位置，控制点 1 由参数 cp1X、cp1Y 确定，控制点 2 由参数 cp2X、cp2Y 确定，终点由参数 *x*、*y* 确定。这条橘黄色的曲线就是从起点连接到终点，由两个控制点联合控制的曲线形状。

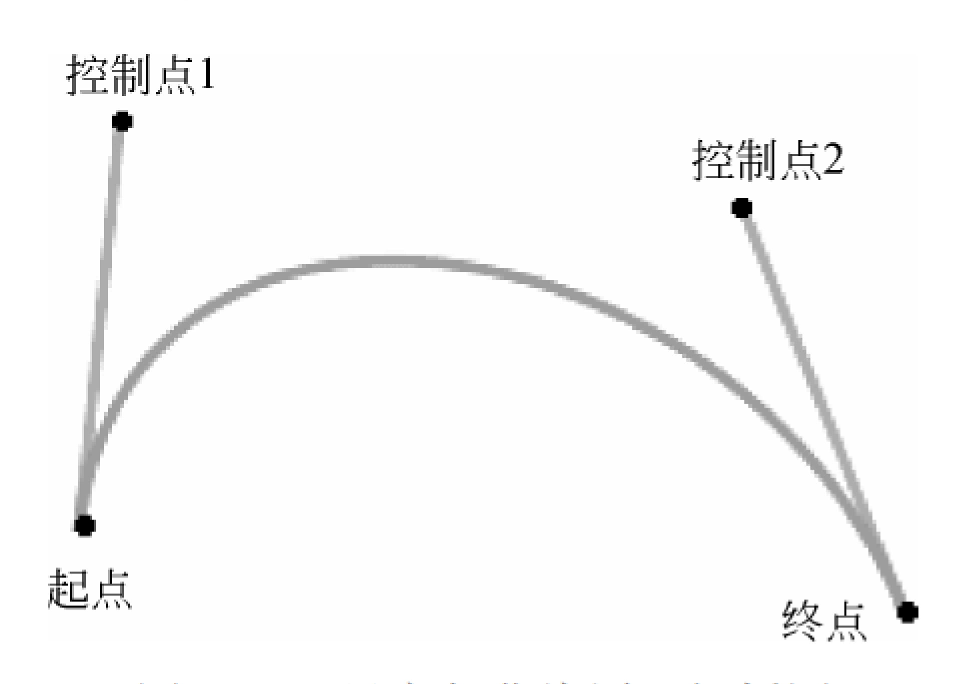


图 8-19　贝塞尔曲线原理解析图

【示例 8-11】 绘制贝塞尔曲线。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //先绘制灰色的辅助线段，宽 2 像素
    context.beginPath();                        //添加第一个子路径
    context.moveTo(100,180);                    //确定当前位置，即绘图起始的位置
    context.lineTo(110,80);                     //直线连接控制点
    context.moveTo(260,100);                    //移动当前位置
    context.lineTo(300,200);                    //直线连接终点
    context.strokeStyle="rgba(0,0,0,0.4)"; //线框颜色为黑色，透明度为 0.4
    context.lineWidth=3;
    context.stroke();
    //再绘制一条曲线，宽 3 像素，线条颜色为橘黄色
    context.beginPath();                        //添加第二个子路径
    context.moveTo(100, 180);                   //确定当前位置，与第一个子路径一致
    context.bezierCurveTo(110, 80, 260, 100, 300, 200);    //确定曲线轮廓
    context.lineWidth = 3;
    context.strokeStyle="rgba(255,135,0,1)";        //线框颜色为橘黄色
    context.stroke();
}
```

运行结果如图 8-20 所示。

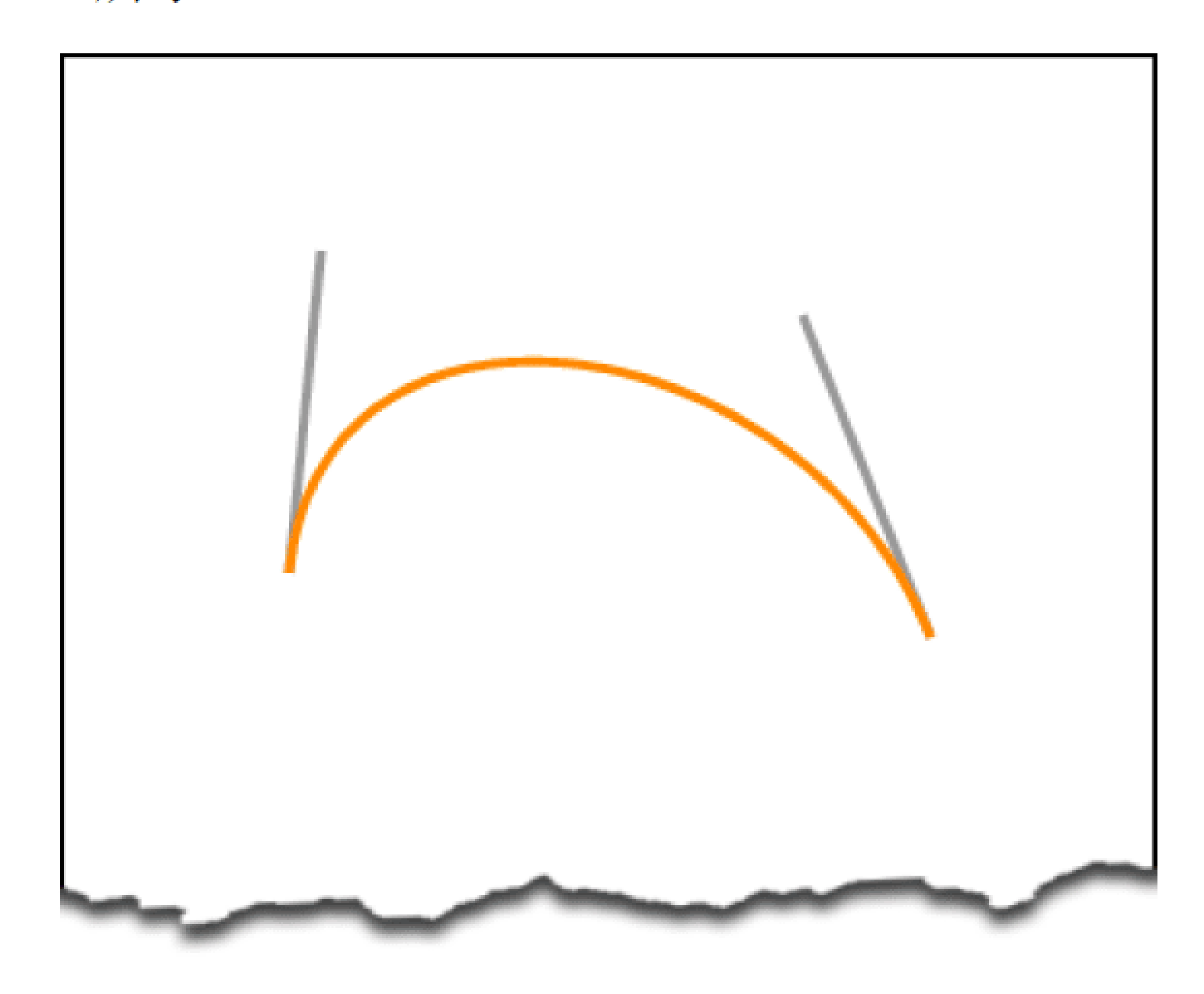


图 8-20　绘制的贝塞尔曲线

代码分析：在绘制辅助线的时候，连接了起点和第一个控制点，第二个控制点和终点。图 8-20 中的黄色线条部分为绘制的曲线部分，连接了曲线的起点和终点，曲线的弯曲形状由其中的两个控制点共同控制。

提示： 关于贝塞尔曲线的变形原理，也已超出我们的研究范围。在这里，只要搞明白如何绘制贝塞尔曲线即可。

8.3.5　使用图像

有的时候，可能需要借助一些现有的图片，使绘图更加灵活和方便。在 Canvas 中，绘

图 API 已经提供了插入图像的方法，只需几行代码就能将图像绘制到画布上。使用 drawImage()方法可将图像添加到 Canvas 画布中，即绘制一幅图像，该方法重载了三次。

（1）把整个图像复制到画布，将其放置到指定点的左上角，并且将每个图像像素映射成画布坐标系统的一个单元。语法如下：

```
drawImage(image, x, y)
```

参数说明：image 表示所要绘制的图像的对象，*x*、*y* 表示要绘制的图像的左上角的位置。

（2）把整个图像复制到画布，但是允许用画布单位来指定想要的图像的宽度和高度。语法如下：

```
drawImage(image, x, y, width, height)
```

参数说明：image 表示所要绘制的图像的对象，*x*、*y* 表示要绘制的图像的左上角的位置，width、height 表示图像所应绘制的尺寸，指定这些参数使得图像可以缩放。

（3）此方法是完全通用的，它允许指定图像的任何矩形区域并复制它，对画布中的任何位置都可进行任何的缩放。语法如下：

```
drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight, destX, destY,
destWidth, destHeight)
```

参数说明：image 表示所要绘制的图像的对象。sourceX、sourceY 表示图像将要被绘制的区域的左上角，这些整数参数用图像像素来度量。sourceWidth、sourceHeight 表示图像所要绘制区域的大小，用图像像素表示。destX、destY 表示所要绘制的图像区域的左上角的画布坐标。destWidth、destHeight 图像区域所要绘制的画布大小。

以上三个方法中的参数 image，都表示所要绘制的图像对象，必须是 Image 对象或 Canvas 元素。一个 Image 对象能够表示文档中的一个<img>标记或者使用 Image()构造函数所创建的一个屏幕外图像。

三种方法中，第一种方法参数最少，所以最简单，但实现的功能有限。第二种方法复杂一点，功能仍然受限。第三种方法，参数最多最复杂，能对图像进行裁剪等操作。在绘图中，根据实际需要，可以从以上三种方法中自由选择。

【示例 8-12】 使用三种方法插入图像。

```
function Draw(){
    var canvas=document.getElementById("canvas");  //获取 canvas 对象
    var context = canvas.getContext("2d");         //获取 2d 上下文绘图对象
    var newImg = new Image();                      //使用 Image()构造函数创建图像对象
    newImg.src= "../images/Chaplin.jpg";           //指定图像的文件地址
    newImg.onload=function(){
        context.drawImage(newImg,0,0);             //从左上角开始绘制图像
        context.drawImage(newImg,250,100,150,200);
                                      //从指定坐标开始绘制图像，并设置图像的宽和高
        context.drawImage(newImg,90,80,100,100,0,0,120,120);
                                      //裁剪一部分图像放在左上角，并稍微放大
    }
}
```

运行结果如图 8-21 所示。

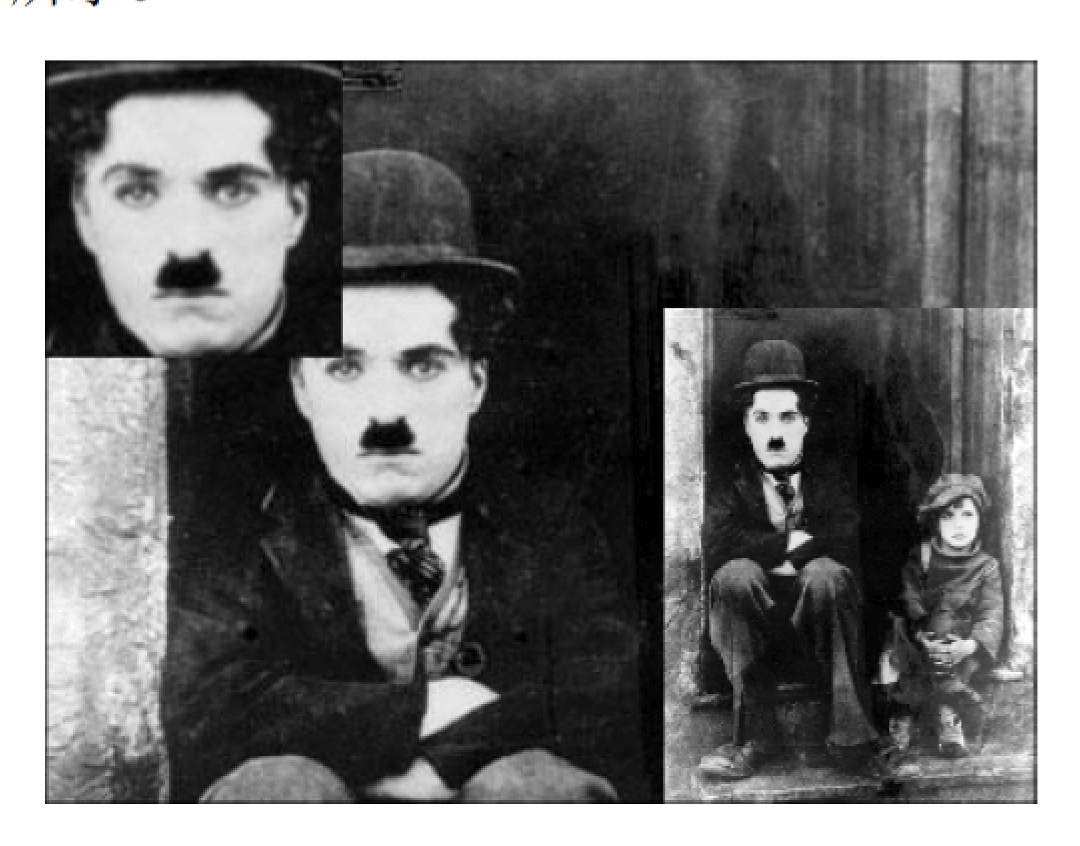


图 8-21　使用三种方法插入图像

代码分析：在示例 8-12 中，使用了三种插入图像的方法。由于参数的个数及表示的意义不同，所以可以灵活运用其特性，选择使用。绘制图像的代码包含在 onload 处理函数中，是因为图像本身需要时间加载，在加载完成之前，图像是不能被绘制的。

图 8-21 中的图像，满画布的图像是用第一种方法绘制的。右下角的图像是用第二种方法绘制的。左上角的头像是用第三种方法绘制的。

提示： 在插入图像之前，需考虑图像加载的时间。如果图像没加载完成就已经执行了 drawImage()方法，则不会显示任何图片。在示例 8-12 中，为图像对象添加了 onload 处理函数，以保证在图像加载完成后执行 drawImage()方法。

8.3.6　剪裁区域

在路径绘图中，我们使用了两大绘图方法，即用于绘制线条的 stroke()方法和用于填充区域的 fill()方法。关于路径的处理，还有一种方法叫做剪裁方法 clip()。

说起剪裁，大多数人会想到剪裁图片，即保留图片的一部分。但是剪裁的实现方法是另一种思维。

比如在火车上，乘客会通过车窗欣赏外面的风景，但会受到车窗的限制，只能看很小的一块区域。外面的风景好比画布，车窗就好比一个裁剪的区域，无论画布里的风景如何绘制，却只能在裁剪区域里表现出来，裁剪区域外是没有任何变化的。也可以理解为，在接下来的绘图中，都是在剪裁区域里进行的。

而裁剪区域是通过路径来确定的。和绘制线条的方法和填充区域的方法一样，也需要预先确定绘图路径，再执行剪裁路径方法 clip()，这样就确定了剪裁区域。剪裁区域的语法如下：

```
clip();
```

该方法没有参数，在设置路径之后执行。

【示例 8-13】 使用裁剪区域绘图。

```
function Draw(){
```

```
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var newImg = new Image();
    newImg.src= "../images/Chaplin.jpg";
    newImg.onload=function(){
        //设置一个圆形的剪裁区域
        ArcClip(context);
        //从左上角开始绘制图像
        context.drawImage(newImg,0,0);
        //设置全局半透明
        context.globalAlpha=0.6;
        //使用路径绘制的矩形
        FillRect(context);
    }
}
//设置一个圆形的剪裁区域
function ArcClip(context){
    context.beginPath();
    context.arc(150,150,100,0,Math.PI*2,true);      //设置一个圆形的绘图路径
    context.clip();                                  //剪裁区域
}
//使用路径绘制的矩形
function FillRect(context){
    context.beginPath();
    context.rect(150,150,90,90);
    context.fillStyle="#f90";
    context.fill();
}
```

运行结果如图 8-22 所示。



图 8-22　使用剪裁区域绘图

代码分析：在绘制图片之前，首先使用方法 ArcClip(context)设置一个圆形剪裁的区域。先设置一个圆形的绘图路径，再调用 clip()方法，即完成了区域的剪裁。接着绘制了图像，把加载的图片，从画布的左上角开始绘制，可以看见，只有在剪裁区域里才有绘制。最后，又使用了路径的方法绘制了矩形，并填充了半透明的颜色，也只在剪裁区域内有绘制。

这说明在裁剪之后的任何绘制，都局限在裁剪区域内部。如果想取消剪裁区域，可以在剪裁区域前先调用 save()方法保存当前上下文状态，在绘制完剪裁图像之后，再调用 restore()方法恢复之前保持的上下文状态，这样就去除了剪裁区域，在接下来的绘图中，就不会被剪裁区域局限了。

对 Draw()函数做一下改动。

【示例 8-14】 取消裁剪区域。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var newImg = new Image();
    newImg.src= "../images/Chaplin.jpg";
    newImg.onload=function(){
        //保存当前状态
        context.save();
        //设置一个圆形的剪裁区域
        ArcClip(context);
        //从左上角开始绘制图像
        context.drawImage(newImg,0,0);
        //恢复被保存的状态
        context.restore();
        //设置全局半透明
        context.globalAlpha=0.6;
        //使用路径绘制的矩形
        FillRect(context);
    }
}
```

运行结果如图 8-23 所示。



图 8-23　使用剪裁区域绘图

代码分析：在剪裁区域之前，首先调用 save()方法保存了当前上下文的状态；在剪裁区域内绘制了图片之后，调用 restore()方法恢复了上下文状态，即剪裁区域之前的状态，所以在接下来的绘图中不再受剪裁区域限制。如图 8-23 中的矩形已经超出了剪裁区域的范围。

提示： *关于方法 save()和方法 restore()，将在后面的章节中进行详细介绍。*

8.3.7　绘制渐变

渐变是一种很普遍的视觉形象，能带来视觉上的舒适感。在 Canvas 中，绘图 API 提供了两个原生的渐变方法，包括线性渐变和径向渐变。渐变，在颜色集上使用逐步抽样的

算法，可以应用在描边样式和填充样式中。使用渐变需要三个步骤：首先是创建渐变对象；其次是设置渐变颜色和过渡方式；最后将渐变对象赋值给填充样式或描边样式。

绘图 API 提供了两种渐变的创建方法：创建线性渐变的 createLinearGradient()方法和创建径向渐变的 createRadialGradient()方法。

1. 创建线性渐变对象

线性渐变是指起始点和结束点之间线性地内插颜色值。创建线性渐变的语法如下：

```
createLinearGradient(xStart, yStart, xEnd, yEnd);
```

参数说明：xStart、yStart 表示渐变的起始点的坐标。xEnd、yEnd 表示渐变的结束点的坐标。返回一个渐变对象。

2. 创建径向渐变对象

径向渐变，是指两个指定圆的圆周之间放射性地插颜色值。创建径向渐变的语法如下：

```
createLinearGradient(xStart, yStart, radiusStart, xEnd, yEnd, radiusEnd);
```

参数说明：xStart、yStart 表示开始圆的圆心坐标。radiusStart 表示开始圆的半径。xEnd、yEnd 表示结束圆的圆心坐标。radiusEnd 表示结束圆的半径。返回一个渐变对象 gradient。

3. 设置渐变颜色和过渡方式

设置渐变颜色，需要在渐变对象上使用 addColorStop()方法，在渐变中的某一点添加一个颜色变化。语法如下：

```
addColorStop(offset, color);
```

参数说明：offset 是一个范围在 0.0 到 1.0 之间的浮点值，表示渐变的开始点和结束点之间的一部分，offset 为 0 对应开始点，offset 为 1 对应结束点。color 是一个颜色值，表示在指定 offset 显示的颜色。

4. 将渐变对象赋值给填充样式或描边样式

描边样式 strokeStyle 和填充样式 fillStyle，都可以使用渐变对象来赋值。当样式被赋值为渐变对象时，绘制出来的描边和填充都会有渐变效果。

【示例 8-15】 绘制线性渐变的矩形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //创建渐变对象：线性渐变
    var grd=context.createLinearGradient(0,0,300,0);
    //设置渐变颜色及方式
    grd.addColorStop(0,"#f90");
    grd.addColorStop(1,"#0f0");
    //将填充样式设置为线性渐变对象
    context.fillStyle=grd;
    context.fillRect(0,0,300,80);
}
```

运行结果如图 8-24 所示。



图 8-24　线性渐变的矩形

代码分析：在示例 8-15 中，按照使用渐变的三个步骤，绘制了线性渐变的矩形。如图 8-24 所示，起始点到结束点的渐变从橘黄色渐变到绿色；起始点到结束点可以确定一条线段，渐变会沿着该线段的垂直方向扩展。设置渐变颜色及过渡方式环节，可以增加使用 addColorStop()方法，以便实现更多颜色的线性渐变。

【示例 8-16】　绘制径向渐变的矩形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //创建渐变对象：径向渐变
    var grd=context.createRadialGradient(50,50,0,100,100,90);
    //设置渐变颜色及方式
    grd.addColorStop(0,"#0f0");
    grd.addColorStop(1,"#f90");
    //将填充样式设置为径向渐变对象
    context.fillStyle=grd;
    context.beginPath();
    context.arc(100,100,90,0,Math.PI*2,true);
    context.fill();
}
```

运行结果如图 8-25 所示。

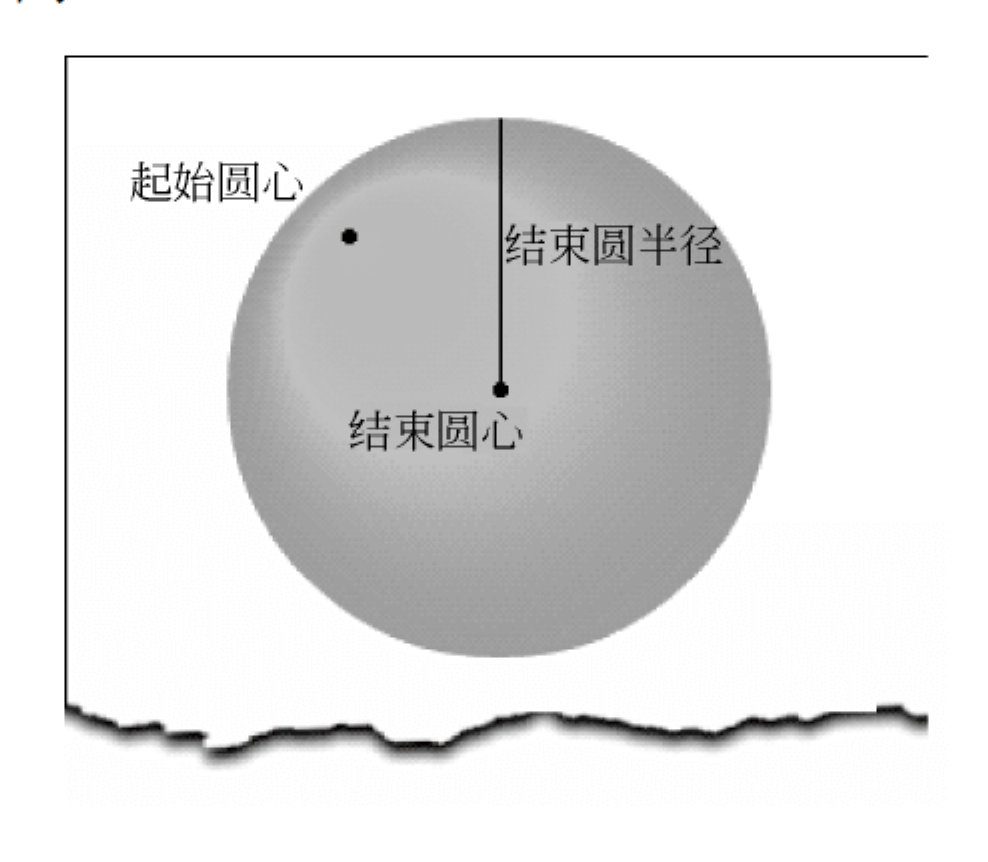


图 8-25　线性渐变的矩形

代码分析：在示例 8-16 中，起始圆的半径为 0，即为一个点。图 8-25 中所示即为起始圆的圆周到结束圆的圆周之间的径向渐变。设置渐变颜色及过渡方式环节，也可以增加使用 addColorStop()方法，以便实现更多颜色的径向渐变。

8.3.8　描边属性

在前面的章节中已经使用过边框样式（即描边样式），相信读者已不再陌生。本节将详细讲解描边过程中使用的各种属性。

描边的过程也是绘制线条的过程，绘制出来的图像是有一定宽度的带有颜色的线条。描边常用的属性除 lineWidth 和 strokeStyle 之外，还包括线条的末端控制属性 lineCap、线条之间的连接属性 lineJoin 和 miterLimit。

1．线条宽度属性lineWidth

描述了画笔（绘制线条）操作的线条宽度，并且这个属性必须大于 0.0。较宽的线条在路径上居中，每边有线条宽的一半。语法如下：

```
lineWidth = [value];
```

参数说明：参数 value 为数字，单位为像素，默认为 1，如图 8-26 所示为 value 值分别是 10、16、20 时的效果。

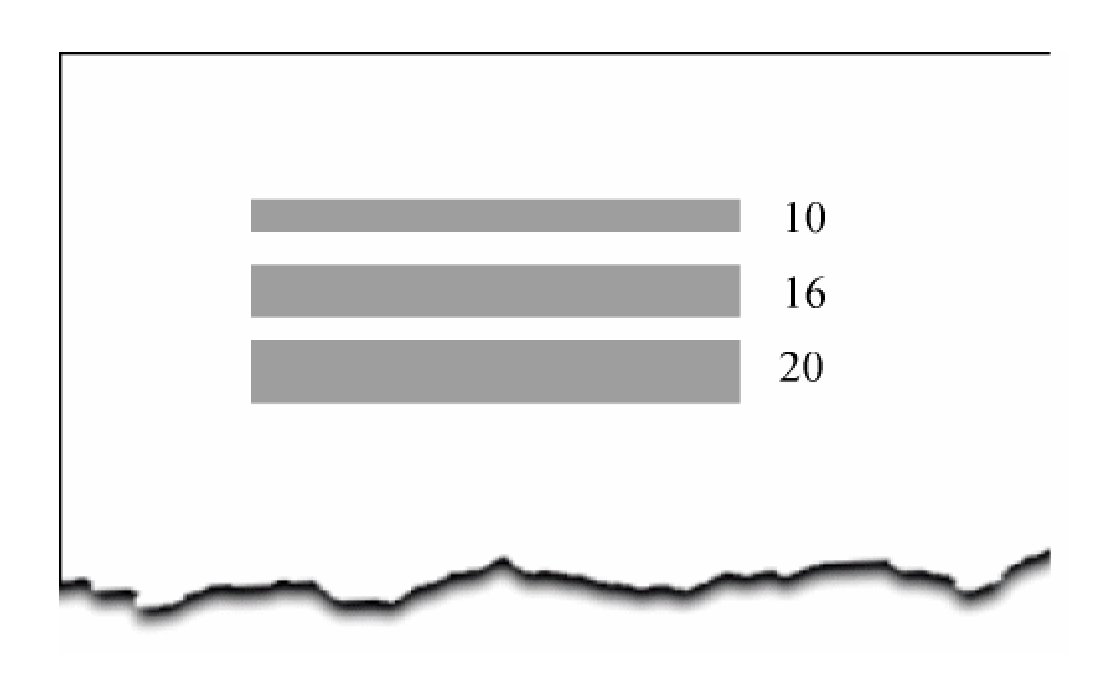


图 8-26　属性 lineWidth 的宽度效果

2．线条样式属性strokeStyle

描述了画笔（绘制线条）操作的线条样式。该样式可以设置为颜色、渐变和模式。语法如下：

```
strokeStyle= [value];
```

参数说明：参数 value 可以设置为字符串表示的颜色，可以是一个渐变对象，也可以是模式对象。

提示： 属性 strokeStyle 和属性 fillStyle 分别用于绘制线条和绘制区域，它们可以接受的值的范围是一样的：颜色、渐变和模式。关于模式将在下一节进行讲述。

3．线帽属性lineCap

描述了指定线条的末端如何绘制。语法如下：

```
lineCap = [value];
```

参数说明：参数 value 的合法值是 butt、round 和 square。默认值是 butt。如图 8-27 所示为 value 值分别为 butt、round 和 square 时的效果。

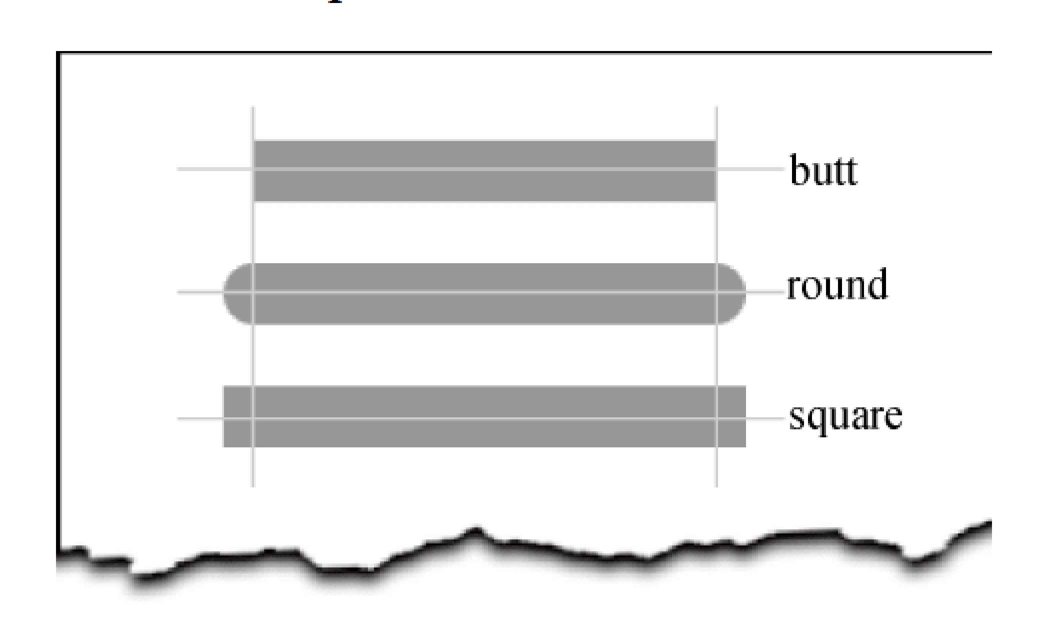


图 8-27　属性 lineCap 各属性值效果

只有当线条具有一定宽度的时候，才能表现出各种值的差异。

- butt：定义了线段没有线帽。线条的末点是平直的而且和线条的方向正交，这条线段在其端点之外没有扩展。
- round：定义了线段的末端为一个半圆形的线帽，半圆的直径等于线段的宽度，并且线段在端点之外扩展了线段宽度的一半。
- square：定义了线段的末端为一个矩形的线帽。这个值和 butt 有着同样的形状效果，但是线段扩展了自己的宽度的一半。

4．线条的连接属性lineJoin

描述了两条线条的连接方式。语法如下：

```
lineJoin = [value];
```

参数说明：参数 value 的合法值是 round、bevel 和 miter。默认值是 miter。如图 8-28 所示为 value 值分别为 round、bevel 和 miter 时的效果。

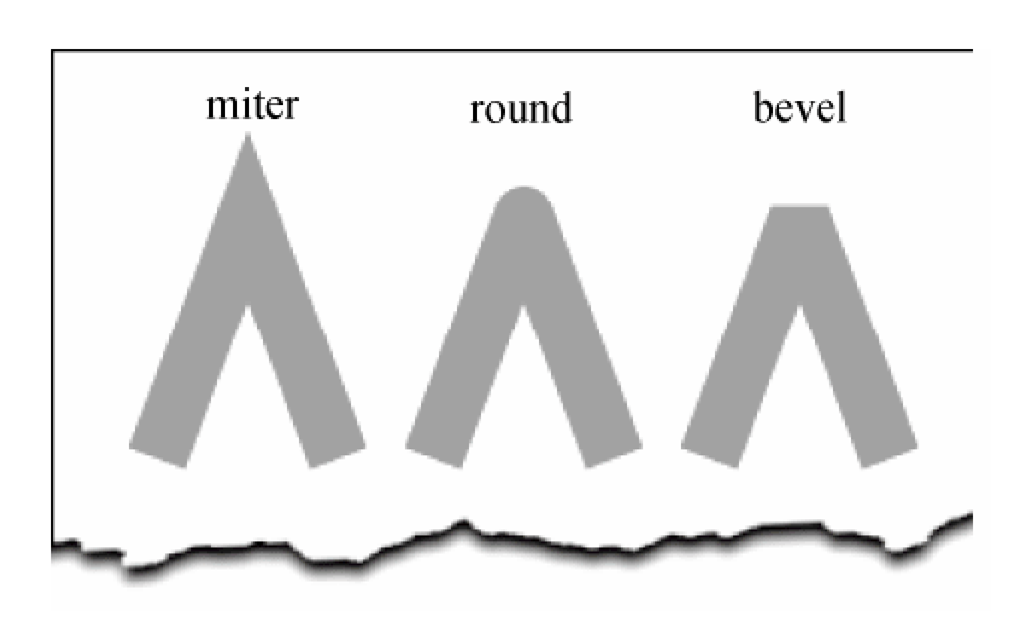


图 8-28　属性 lineJoin 各属性值效果

当一个路径包含了线段或曲线相交的交点的时候，lineJoin 属性可以表现这些交点的连接方式。不过只有当线条较宽的时候，才能表现出不同连接方式的差异。

- miter：定义了两条线段的外边缘一直延伸到它们相交。当两条线段以一个锐角相交时，连接的地方可能会延伸到很长。
- round：定义了两条线段的外边缘应该和一个填充的弧接合，这个弧的直径等于线段的宽度。

- bevel：定义了两条线段的外边缘应该和一个填充的三角形相交。

5. 扩展的线条连接属性miterLimit

进一步描述了如何绘制两条线段的交点。语法如下：

```
miterLimit= [value];
```

参数说明：参数 value 为数值。

当宽线条的 lineJoin 属性为 miter 时，并且两条线段以锐角相交的时候，连接的地方可能会相当长。miterLimit 属性可以为该延伸的长度设置一个上限。这个属性表示延伸的长度和线条长度的比值。默认是 10，表示延伸的长度不应该超过线条宽度的 10 倍。如果延伸的长度超过这个长度，就变成斜角了。当属性 lineJoin 的值为 round 或 bevel 的时候，属性 miterLimit 是无效的。

8.3.9　模式

模式是一个抽象的概念，描述的是一种规律。在 Canvas 中，通常会为贴图图像创建一个模式，用于描边样式和填充样式，可以绘制出带图案的边框和背景图。在 Canvas 中，模式是一个对象，使用 createPattern()方法可以为贴图图像创建一个模式，语法如下：

```
createPattern(image, repetitionStyle)
```

参数说明：image 描述了一个贴图图像，可以是一个图像对象，也可以是一个 Canvas 对象。repetitionStyle 描述了该贴图图像的循环平铺方式，有 4 个值分别为：repeat、repeat-x、repeat-y 和 no-repeat。repeat 表示图像在各个方向上循环平铺；repeat-x 表示图像在横向上循环平铺；repeat-y 表示图像在纵向上循环平铺；no-repeat 表示图像只使用一次，不平铺。

【示例 8-17】 用贴图模式填充矩形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var img = new Image();              //使用 Image()构造函数创建图像对象
    img.src = '../images/flower.gif';   //指定图像的文件地址
    img.onload = function(){
        var ptrn = context.createPattern(img,'repeat');
                                        //创建一个贴图模式，循环平铺图像
        context.fillStyle = ptrn;       //设置填充样式为贴图模式
        context.fillRect(0,0,300,200);  //填充矩形
    }
}
```

运行结果如图 8-29 所示。

代码分析：使用贴图模式的代码包含在 onload 处理函数中，是因为图像本身需要时间加载，在加载完成之前，创建出来的贴图模式是无效的。贴图模式也可以用于描边样式。

提示：模式可用于背景的绘制，使用方法与 CSS 中的背景样式很相像。其中循环平铺方式 repeat-x、repeat-y 在部分浏览器中支持得不是很好，如在 Firefox 中。

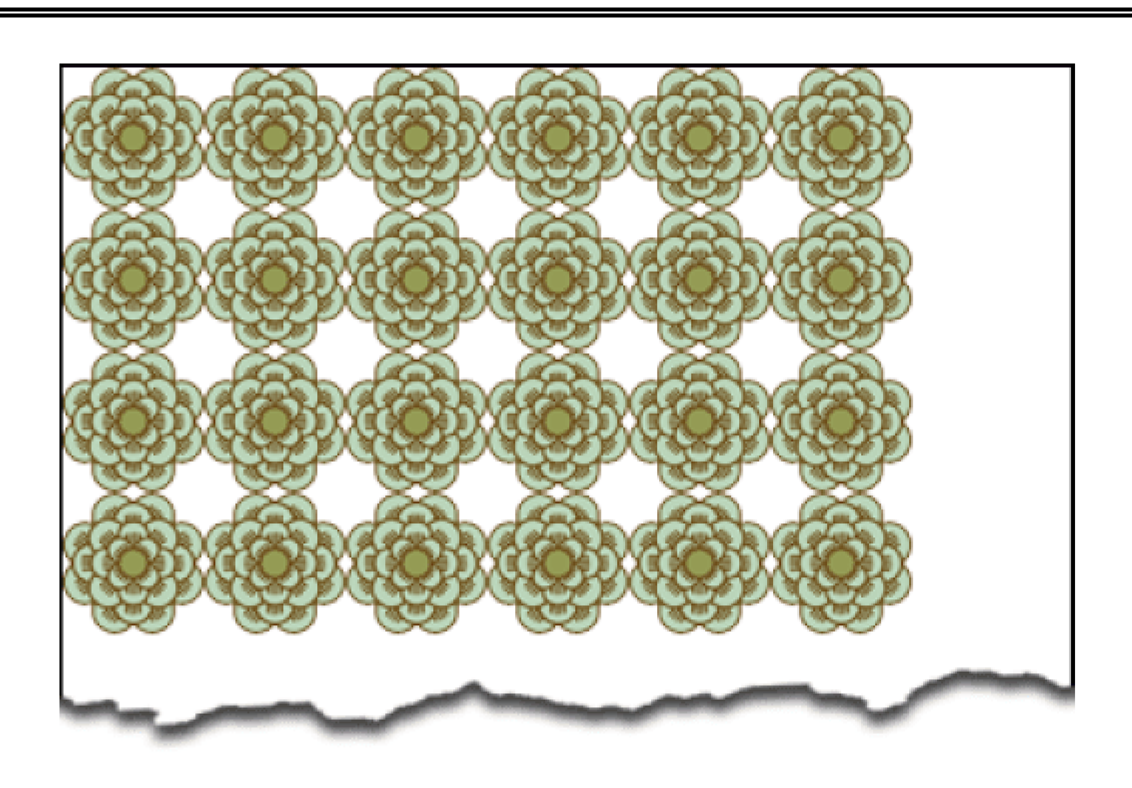


图 8-29　使用贴图模式填充的矩形

8.3.10　变换

在绘制图形的过程中，如果一种形状的图形要绘制多次，显然增加了复杂性。Canvas 绘图 API 提供了多种变换方法，为实现复杂的绘图操作提供了便捷的方法。常见的变换的方法包括平移、缩放、旋转和变形等。

在默认情况下，Canvas 的坐标空间是以左上角(0,0)作为原点，*x* 值向右增加，*y* 值向下增加，坐标空间中的一个单位通常转换为像素。也就是说，坐标空间默认包含了一些基本属性。所以，可以把变换理解为改变了坐标空间的一些属性设置。

接下来通过案例来讲解图像的变换。

1. 移动变换

移动变换是将整个坐标系统设置一定的偏移数量，绘制出来的图像也会跟着偏移。为坐标系统添加水平的和垂直的偏移实现移动。语法如下：

```
translate(dx,dy);
```

参数说明：dx 为水平方向上的偏移量；dy 为垂直方向上的偏移量。添加偏移后，会将偏移量附加给后续的所有坐标点。

使用移动的方法，将绘制的圆形脸谱移动到合适的位置。

【示例 8-18】 绘制一个圆形脸谱。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //设置移动偏移量
    context.translate(200,120);
    //绘制一个圆形脸谱
    ArcFace (context);
}
function ArcFace(context){
    //绘制一个圆形边框
    context.beginPath();
    context.arc(0,0,90,0,Math.PI*2,true);
    context.lineWidth=5;
    context.strokeStyle="#f90";
```

```
    context.stroke();
    //绘制一个脸谱
    context.beginPath();
    context.moveTo(-30,-30);
    context.lineTo(-30,-20);
    context.moveTo(30,-30);
    context.lineTo(30,-20);
    context.moveTo(-20,30);
    context.bezierCurveTo(-20, 44, 20, 30, 30, 20);
    context.strokeStyle="#000";
    context.lineWidth=10;
    context.lineCap="round";
    context.stroke();
}
```

运行结果如图 8-30 所示。

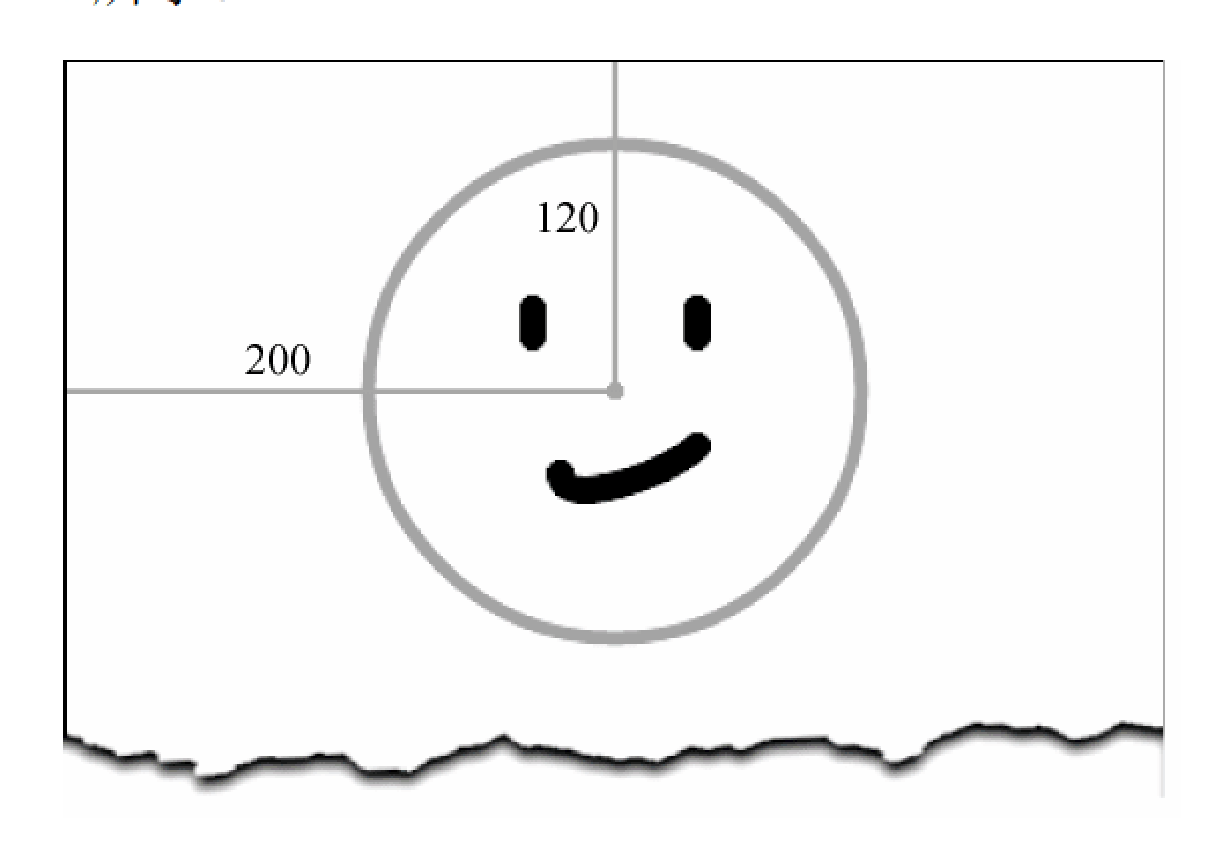


图 8-30　绘制的圆形脸谱

代码分析：使用函数 ArcFace()绘制一个圆形脸谱的图像。该图像是以原点为中心绘制的，由于在绘图开始之前，已经将坐标系统进行了偏移设置，所以绘制的所有坐标都进行相应的坐标偏移。其值，在 *X* 坐标方向上的偏移量为 200，*Y* 坐标方向上的偏移量为 120。

如果需要调整图像的位置，只需调整坐标系统的偏移量就可以了，不用再在新的位置重新绘图，很直观地实现了图像的移动。

2．缩放变换

缩放变换是将整个坐标系统设置一对缩放因子，绘制出来的图像会相应地缩放。为坐标系统添加一个缩放变换，设置独立的水平和垂直缩放因子实现图像的缩放。语法如下：

```
scale (sx,sy);
```

参数说明：sx 为水平方向上的缩放因子；sy 为垂直方向上的缩放因子。sx 和 sy 为大于 0 的数字，当其值大于 1 时，为放大图像，小于 1 时，为缩小图像。

使用缩放的方法，可以将绘制的圆形脸谱变换成椭圆形。

【示例 8-19】 将圆形脸谱变换成椭圆形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    context.translate(200,120);
```

```
    //缩放图像，在水平方向和垂直方向设置不同的缩放因子
    context.scale(0.6,0.4);
    //绘制一个圆形脸谱
    ArcFace (context);
}
```

运行结果如图 8-31 所示。

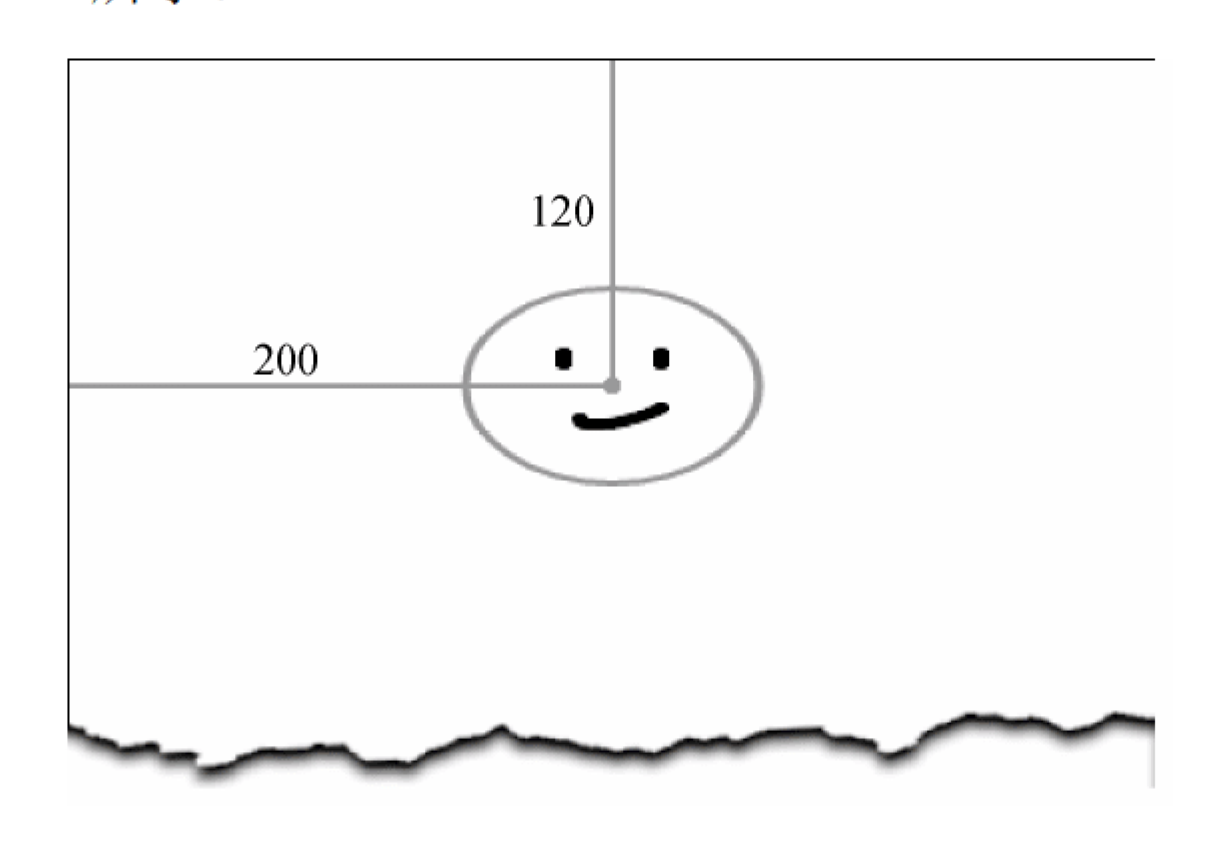


图 8-31　缩放后的椭圆形脸谱

代码分析：示例 8-19 中使用的函数 ArcFace ()绘制的仍然是一个圆形脸谱。在绘制之前，为坐标系统添加了缩放变换，在 *X* 坐标方向上缩小为实际的 0.6 倍，*Y* 坐标方向上缩小为实际的 0.4 倍。缩放后平移到适当的位置，效果如图 8-31 所示，圆形脸谱变成了椭圆形。

提示： 示例 8-19 中使用的函数 ArcFace ()即是示例 8-18 中的函数 ArcFace ()，不再重复列出代码。

通过缩放变换，可以改变图像在 *X* 方向和 *Y* 方向上的比例，丰富图像的表现。

3．旋转变换

旋转变换是将整个坐标系统设置一个旋转的角度，绘制出来的图像会相应地旋转。为坐标系统指定一个旋转的弧度，绘制出来的图像也会做相应的旋转，即实现了图像的旋转变换。语法如下：

```
rotate(angle);
```

参数说明：angle 旋转的量用弧度表示。正值表示顺时针方向旋转，负值表示逆时针方向旋转。旋转的中心点为坐标系统的原点。

提示： 这里的旋转量是用弧度表示的。如需把角度转换为弧度，请乘以 Math.PI 并除以 180。

使用旋转的方法，可以将绘制的圆形脸谱进行倾斜。

【示例 8-20】 将圆形脸谱变换成椭圆形。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    context.translate(200,120);
```

```
    //旋转图像，顺时针旋转 30 度
    context.rotate(Math.PI/6);
    context.scale(0.6,0.4);
    //绘制一个圆形脸谱
    ArcFace (context);
}
```

运行结果如图 8-32 所示。

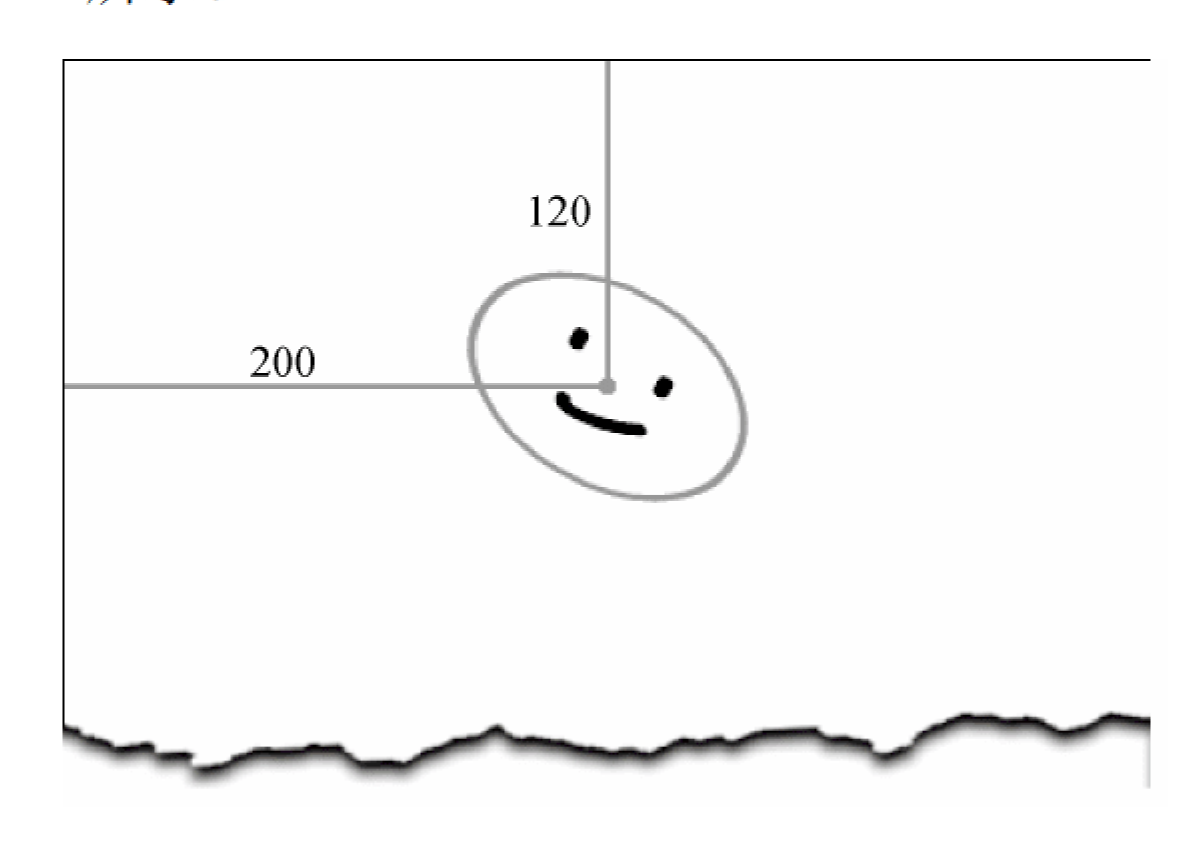


图 8-32　旋转后的脸谱

代码分析：示例 8-20 中使用的函数 ArcFace()绘制的仍然是一个圆形脸谱。在绘制之前，为坐标系统又添加了旋转变换，沿顺时针方向旋转 30 度。旋转后的效果如图 8-32 所示，椭圆形脸谱倾斜了。

4．矩阵变形

通过使用矩阵，可以让图形变形更加复杂。在默认绘图的坐标系统中，事实上存在一个默认的矩阵，当我们对这个矩阵进行修改时，就会造成图形的变形。矩阵变形的语法如下：

```
transform(m11,m12,m21,m22,dx,dy);
```

参数说明：该方法中的 6 个参数组成一个变形矩阵，与当前矩阵进行乘法运算，形成新的矩阵系统。该变形矩阵的形式如下：

```
m11    m21    dx
m12    m22    dy
0       0     1
```

关于详细的矩阵变形原理，需要掌握矩阵的相关知识，具体可参考数学及图形学相关资料。不过这里可以先通过几个特例了解其大概的使用方法。

前面已经讲过三种变换，分别是移动、缩放和旋转。这些变换相对容易理解，其实都可看作矩阵变形的特例。

- ❑ 移动 translate(dx,dy)，也可以使用 transform(1,0,0,1,dx,dy)或 transform(0,1,1,0,dx,dy)来实现。
- ❑ 缩放 scale(sx,sy)，也可以使用 transform(sx,0,0,sy,0,0)或 transform(0,sy,sx,0,0,0)来实现。

- 旋转 rotate(A)，也可以使用 transform(cosA,sinA,-sinA,cosA,0,0)或 transform(-sinA, cosA,cosA,sinA,0,0)来实现。

当然，也可以通过 transform()方法实现更加复杂的变形，可以参考数学及图形学相关资料，这里不再详细讲解。

说明：所有的变换都是以原点为基点进行的。所以绘制的图像最好以原点为中心，然后再进行变换，否则图像位置会变得难以控制。

8.3.11　使用文本

在 Canvas 中，也可以绘制文本。可以使用填充的方法绘制，也可以使用描边的方法绘制，在绘制文字之前，还可以设置文字的字体样式和对其方式。绘制文本有两个方法，分别是填充绘制方法 fillText()和描边绘制方法 strokeText()。语法如下：

```
fillText(text,x,y,maxwidth);
strokeText(text,x,y,maxwidth);
```

参数说明：参数 text 表示要绘制的文本。参数 *x* 表示绘制文本的起点横坐标。参数 *y* 表示绘制文本的起点纵坐标。参数 maxwidth 为可选参数，表示显示文本的最大宽度，可以防止文本溢出。

在绘制文本之前，可以先对文本进行样式设置。绘图 API 提供了专门用于设置文本样式的属性，可以设置文本的字体、大小等，类似于 CSS 的字体属性。也可以设置对齐方式，包括水平方向上的对齐和垂直方向上的对齐。文本的相关属性如表 8.4 所示。

表 8.4　文本的相关属性

属　　性	值	说　　明
font	CSS 字体样式字符串	设置字体样式
textAlign	start | end | left | right | center	设置水平对齐方式，默认为 start
textBaseline	top | hanging | middle | alphabetic | ideographic | bottom	设置垂直对齐方式，默认为 alphabetic

【示例 8-21】 绘制文本示例。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //填充方式绘制文本
    context.fillStyle="#f90";
    context.font="bold 36px impact";
    context.fillText("Hello World!",10,50);
    //描边方式绘制文本
    context.strokeStyle="#f90";
    context.font="bold italic 36px impact";
    context.strokeText("Hello World!",10,100);
}
```

运行结果如图 8-33 所示。

图 8-33　绘制的文本

代码分析：示例 8-21 中，font 属性设置了文本样式：字体为 impact、加粗效果 bold、文字大小为 30px、倾斜效果 italic。其填充样式仍然使用 fillStyle 来设置，描边样式仍然使用 strokeStyle 来设置。

有时候需要知道绘制的文本宽度，以方便布局。绘图 API 提供了这样的方法 measureText()，用来获取文本的宽度。语法如下：

```
measureText (text);
```

参数说明：参数 text 表示所要绘制的文本。该方法会返回一个 TextMetrics 对象，表示文本的空间度量。可以通过该对象的 width 属性获取文本的宽度。

【示例 8-22】 度量的绘制文本。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var txt="Hello World!";
    //填充方式绘制文本
    context.fillStyle="#f90";
    context.font="bold 30px impact";
    //根据已经设置的文本样式度量文本
    var tm=context.measureText(txt);
    context.fillText(txt,10,50);
    context.fillText(tm.width,tm.width+15,50);
    //描边方式绘制文本
    context.strokeStyle="#f90";
    context.font="bold italic 36px impact";
    //根据已经设置的文本样式度量文本
    tm=context.measureText(txt);
    context.strokeText(txt,10,100);
    context.strokeText(tm.width,tm.width+15,100);
}
```

运行结果如图 8-34 所示。

图 8-34　度量文本的数值

代码分析：度量文本是以当前设置的文本样式为基础的。即文本样式确定以后，即可获取文本的度量，不需要等待绘制文本完成后再去度量。

8.3.12　阴影效果

阴影效果可以增加图像的立体感。为图像添加阴影效果，可利用绘图 API 提供的绘制阴影的属性。阴影属性不会单独去绘制阴影，只需要在绘制任何图像之前，添加阴影属性，就能绘制出带有阴影效果的图像。设置阴影的属性有 4 个，如表 8.5 所示。

表 8.5　阴影属性

属　　性	值	说　　明
shadowColor	符合 CSS 规范的颜色值	可以使用半透明颜色
shadowOffsetX	数值	阴影的横向位移量，向右为正，向左为负
shadowOffsetY	数值	阴影的纵向位移量，向下为正，向上为负
shadowBlur	数值	高斯模糊，值越大，阴影边缘越模糊

【示例 8-23】 文字和图形添加阴影。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //设置阴影属性
    context.shadowColor="#666";
    context.shadowOffsetX=5;
    context.shadowOffsetY=5;
    context.shadowBlur=5.5;
    //绘制文本
    context.fillStyle="#f90";
    context.font="bold 36px impact";
    context.fillText("Hello World!",10,50);
    //路径绘制图形
    context.fillStyle="#f90";
    context.arc(100,100,30,0,Math.PI*2,false);
    context.fill();
}
```

运行结果如图 8-35 所示。



图 8-35　文字和图形的阴影效果

代码分析：在示例 8-23 中，在绘制文本和图形之前，设置了阴影属性。其后绘制的文本和图形均附带阴影效果。

提示：阴影属性可以应用于任何绘制的图像中，也包括图片。

8.3.13　状态的保存与恢复

在剪裁区域一节中，介绍了去除剪裁区域的方法，即通过保存和恢复绘图状态来实现。在绘图过程中，绘图状态可能会不断改变，如果某个状态需要多次使用，可以保存这个状态，待需要的时候，再把这个状态恢复。

绘图 API 提供了状态保存方法 save()和状态恢复方法 restore()，分别用于绘图状态的保存与恢复。使用方法比较简单，语法如下：

```
save();
restore();
```

状态的保存和恢复是通过数据栈进行的。当调用 save()方法时，当前的状态会保存到一个数据栈里；当调用 restore()方法时，会取出最后一次保存到数据栈里的数据，即恢复最后一次保存的状态。

提示：栈是一种数据结构，是按照后进先出的原则来存储数据的。这好比打开一个箱子，先拿出来最上面的东西，也就是最后放进去的东西，遵循后进先出的原则。

其中绘图的状态由以下几个因素确定。

（1）坐标系统的变换：平移、缩放、旋转和矩阵变形。

（2）绘图 API 提供的所有属性：globalAlpha、globalCompositeOperation、strokeStyle、fillStyle、lineWidth、lineCap、lineJoin、miterLimit、font、textAlign、textBaseline、shadowOffsetX、shadowOffsetY、shadowBlur、shadowColor 等。

（3）剪裁的区域。

状态包含的内容仅局限于以上几点内容，至于当前绘制出来的图形或路径，不属于状态内容，也不会被保存或恢复。

【示例 8-24】 状态的保存与恢复。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    //设置填充颜色为绿色
    context.fillStyle = "#0f0";
    //保存状态
    context.save();
    //设置新的填充颜色为橘黄色
    context.fillStyle = "#F90";
    //填充一个矩形区域
    context.beginPath();
    context.rect(10,10,90,90);
    context.fill();
    //恢复状态
    context.restore();
    //填充一个圆形区域
    context.beginPath();
    context.arc(100,100,50,0,Math.PI*2,true);
```

```
    context.fill();
}
```

运行结果如图 8-36 所示。

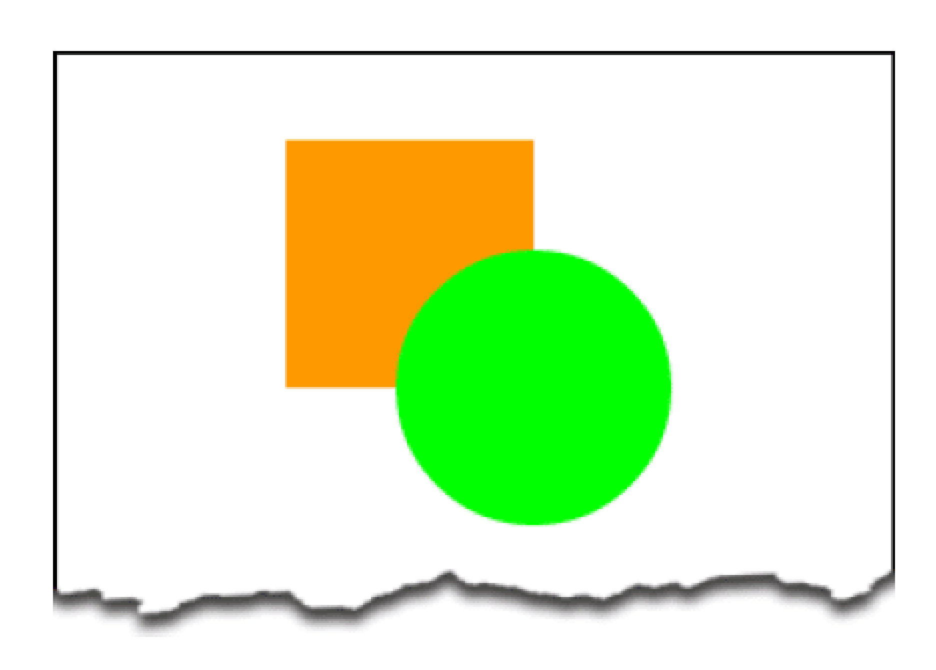


图 8-36　状态的保存于恢复

代码分析：示例 8-24 中，首先将属性 fillStyle 设置为绿色，并调用 save()方法保存状态；接着再将属性 fillStyle 设置为橘黄色，并填充矩形；最后调用 restore()方法恢复状态，并填充一个圆形。如图 8-36 所示，圆形被填充为绿色。

提示：状态的保存与恢复，是用来保存和恢复当时绘图的状态环境，保存状态之后绘制出来的图形，不会因为状态的恢复而消失，即恢复的不是绘制的内容，请准确理解。

8.3.14　操作像素

在 Canvas 中，绘图 API 还提供了像素级的操作方法，分别是：createImageData()、getImageData()和 putImageData()。使用这些方法，可以直接操纵底层的像素数据。这里还使用了一个图像数据对象 ImageData。

1. ImageData对象

在处理像素数据的过程中，该对象作为一种处理的媒介，保存了可以操作的图像像素数据。细致的图像操作，就是在对象 ImageData 中进行的。

该对象有三个属性：width、height 和 data。其中 width 表示每行有多少个像素；height 表示有多少行像素；data 是一个一维数组，保存了所有像素的颜色值，按照从左到右、从上到下的顺序依次存储。

关于颜色值：每个像素的颜色值包含 4 个数字，分别代表红、绿、蓝和透明度，各个数字值的范围均为 0 至 255，包括透明度的值也是这个范围。

2. 获取图像数据的方法getImageData()

从 Canvas 上下文中获取图像数据。语法如下：

```
getImageData(sx,sy,sw,sh);
```

参数说明：该方法有 4 个参数。sx、sy 分别表示所获取区域的起点横坐标和起点纵坐

标；sw、sh 分别表示所获取区域的宽度和高度；返回的结果是一个 ImageData 对象。

3．绘制图像数据方法putImageData()

将处理好的图像数据绘制到 Canvas 中。语法如下：

```
getImageData(imagedata,dx,dy[,dirtyX,dirtyY,dirtyWidth,dirtyHeight]);
```

参数说明：该方法有 3 个必需参数和 4 个可选参数。imagedata 为 ImageData 对象，包含了图像数据；dx、dy 分别表示绘制的起点横坐标和起点纵坐标；可选参数 dirtyX、dirtyY、dirtyWidth 和 dirtyHeight 确定了一个以 dx 和 dy 为坐标原点的矩形，分别表示矩形的起点横坐标、起点纵坐标、宽度和高度。如果加上这 4 个参数，绘制的图像仅限制在该矩形范围内，类似一个裁剪的区域。

4．创建图像数据的方法createImageData()

直接创建一组空的图像数据。语法如下：

```
createImageData(sw,sh);
```

参数说明：该方法有两个参数。sw 表示图像数据的宽度；sh 表示图像数据的高度；创建的结果是返回一个 ImageData 对象。

提示：不是所有浏览器都实现了 createImageData，所以使用的时候，注意把握。

【示例 8-25】 图像底片效果。

```
function Draw(){
    var canvas=document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var newImg = new Image();
    newImg.src= "../images/Chaplin.jpg";
    newImg.onload=function(){
        context.drawImage(newImg,0,0,400,300);
        context.save();
        //获取图像数据
        var imageData = context.getImageData(0, 0, 400, 300);
        //修改 ImageData 对象的 data 数据，处理为反向
        for(var i=0,n=imageData.data.length;i<n;i+=4){
            //红色部分
            imageData.data[i+0] =255-imageData.data[i+0];
            //绿色部分
            imageData.data[i+1] =255-imageData.data[i+1];
            //蓝色部分
            imageData.data[i+2] =255-imageData.data[i+2];
        }
        //绘制该图像数据
        context.putImageData(imageData,200,150);
    }
}
```

运行结果如图 8-37 所示。

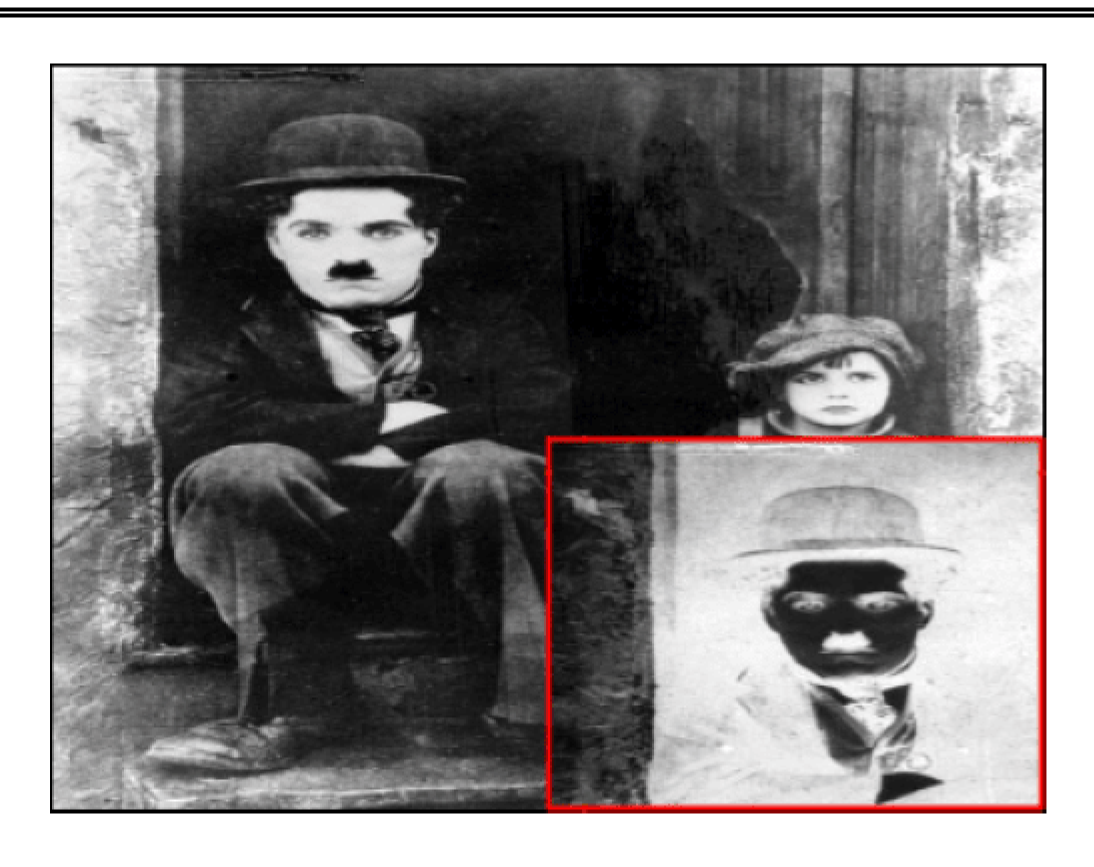


图 8-37　图像的底片效果

代码分析：在示例 8-25 中实现底片效果，分为三步骤。首先，获取整个图像；接着对图像中的每个像素进行反向处理；最后将图像数据再次绘制到指定的位置。

提示：示例 8-25 在本地直接调试可能出错。这是由于 JavaScript 的同源策略对 context.getImageData 的影响。该策略是基于浏览器的安全，擅自禁用该策略可能会造成安全隐患。你可以在本地建立一个站点，将该示例放在站点中运行。

直接操作像素，通过 ImageData 可以完成很多功能。除了上面的示例的底片效果，还可以实现诸如图像滤镜、数学可视化（如分形和其他特效）等效果。

8.4　实验室：在 Canvas 中实现动画

在 Canvas 中，除了一些常规的绘图，还能开发一些动画甚至是游戏，而这一切动画的实现，除了充分利用 Canvas 绘图 API，还需要 JavaScript 的鼎力相助。

在 Canvas 中，动画是通过一系列连续的画面按顺序呈现来实现的。而这些连续的画面是即时绘制出来的，为了使动画更加流畅，可能在很短的时间内重新绘制很多次。动画的大致实现流程可以分以下几个步骤。

（1）清空画布。

（2）改变绘图的状态，包括坐标系统变换、各种属性的修改等。

（3）重新绘制图形。

（4）回到第一步。

在上面几个步骤的轮回中，需要将绘制动作放在一个定时器里。Javascript 提供了两个定时器方法，分别是：setInterval(code,millisec)和 setTimeout(code,millisec)。关于这两个方法，可以去查阅 Javascript 相关资料，这里不再讲解。

为了方便绘制图像，可能需要频繁修改绘图的状态，及时保存和恢复状态，可以让你方便许多。

1．动画：制作一个碰碰球

现在就利用学过的绘图 API 知识，来绘制一个简单的碰碰球。在这个示例中，球在画

布中不停地移动，当碰到边界时，改变球的颜色，并弹回继续移动，如此循环下去。同时给出“开始”和“暂停”按钮，以便在 Canvas 画布之外实现控制，最终效果如图 8-38 所示。

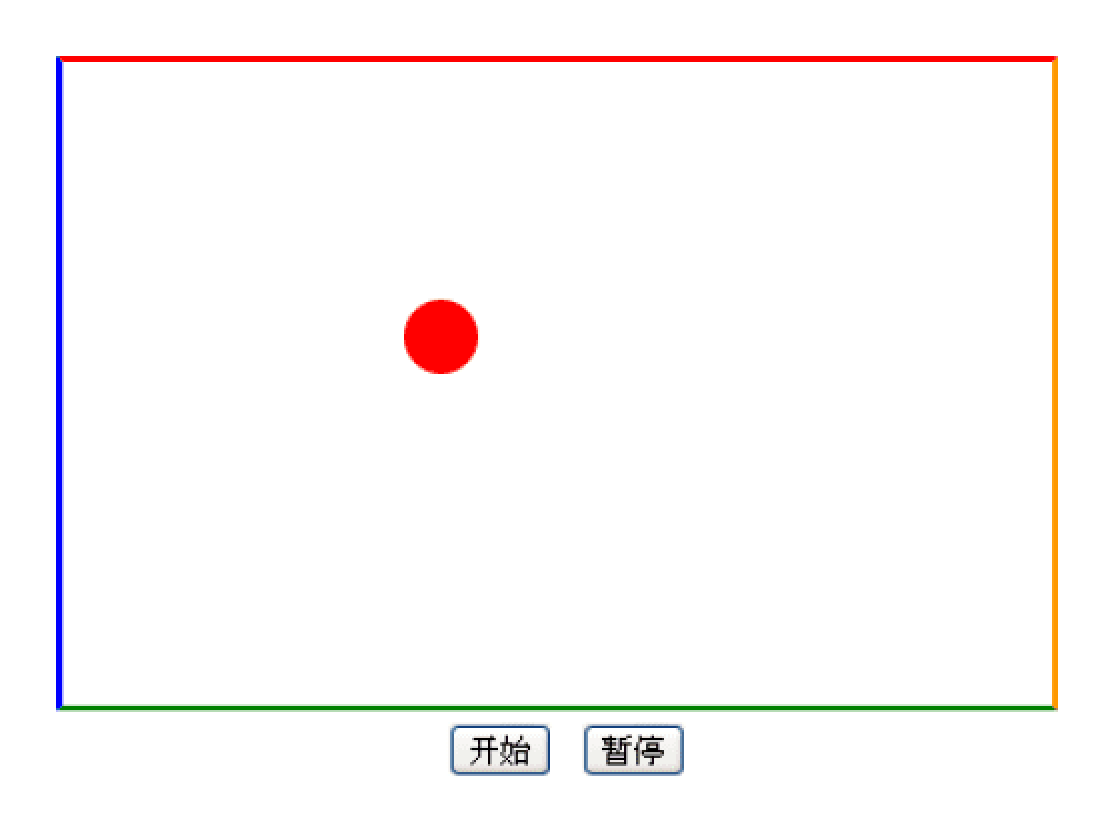


图 8-38　碰碰球动画应用

下面我们就一步一步地来实现它。

当球碰到上面时，会变成红色；碰到右边时会变成橘黄色；碰到下面时会变成绿色；碰到左边时会变成蓝色。为方便视觉识别，下面会将画布的 4 个边框设置成对应的颜色，并设置两个像素的宽度，以便更加醒目。这个环节我们用 CSS 来实现，代码如示例 8-26 所示。

【示例 8-26】 碰碰球的 Canvas 元素。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>canvas 动画--碰碰球</title>
<style type="text/css">
canvas {
    border-top:2px solid #f00;
    border-right:2px solid #f90;
    border-left:2px solid blue;
    border-bottom:2px solid green;
}
</style>
<script type="text/javascript">
//此处添加 JavaScript 代码
</script>
</head>
<body>
<div align="center">
<canvas id="canvas" width="400" height="300">你的浏览器不支持该功能！
</canvas><br />
<input onclick="animation.start()" value="开始" type="button">
<input onclick="animation.pause()" value="暂停" type="button"></div>
</body>
</html>
```

接下来开始在头部添加 JavaScript 代码。首先定义一个动画对象，并为该对象添加一些属性。这里的属性是根据应用需要自定义的。主要包括：定时器、*X* 方向偏移量、*Y* 方

向偏移量、*X* 方向移动步长、*Y* 方向移动步长、圆形半径、圆形的填充颜色及动画的间隔时间（毫秒），代码如示例 8-27 所示。

【示例 8-27】 定义动画对象并添加系列属性。

```
//定义一个动画对象
var animation={};
//为该对象添加属性
//添加定时器
animation.interval=null;
//移动变换 x 方向的偏移量
animation.x=100;
//移动变换 y 方向的偏移量
animation.y=50;
//x 方向的移动步长
animation.xstep=2;
//y 方向的移动步长
animation.ystep=2;
//圆形半径
animation.radius=15;
//填充圆形的颜色
animation.color="red";
//动画间隔时间:毫秒
animation.delay=10;
```

接下来为动画对象添加 update()方法。该方法用来更新偏移量 *x* 和 *y*，还有填充的颜色值 color。因为该方法和上面添加的属性，同属于动画对象 animation，所以在 update()方法中，可以通过 this 来引用 animation 的各个属性。

update()方法会检测各个边缘，并在球碰到边缘的时候改变其颜色。该方法有两个参数 width 和 height，分别为画布的宽和高。代码如示例 8-28 所示。

【示例 8-28】 为动画对象添加 update()方法。

```
//更新偏移量 x 和 y
animation.update=function(width,height){
    //改变 x 坐标
    this.x+=this.xstep;
    this.y+=this.ystep;
    //左边缘检测
    if(this.x<this.radius){
        this.x=this.radius;
        this.xstep=-this.xstep;
        this.color="blue";
    }
    //右边缘检测
    if((this.x+this.radius)>width){
        this.x=width-this.radius;
        this.xstep=-this.xstep;
        this.color="#f90";
    }
    //上边缘检测
    if(this.y<this.radius){
        this.y=this.radius;
        this.ystep=-this.ystep;
        this.color="red";
    }
    //下边缘检测
```

```
    if((this.y+this.radius)>height){
        this.y=height-this.radius;
        this.ystep=-this.ystep;
        this.color="green";
    }
};
```

接下来添加一个绘图方法 draw()。在绘图之前，会首先清空画布，再调用 animation 对象的 update()方法，来更新动画属性。也就是前面提到的制作动画的第一步和第二步，清空画布和变更绘图状态。

在清空画布之前，先对状态进行保存；绘制完成后，会及时恢复绘图状态，为下一次绘图做好准备。

圆形的填充样式设置为动画对象的 color 属性，而 color 属性在边缘检测的时候是会被改变的。绘图方法如示例 8-29 所示。

【示例 8-29】 为动画对象添加 draw()方法。

```
//绘制图形
animation.draw=function(){
    var canvas=document.getElementById("canvas");
    var width=canvas.getAttribute("width");
    var height=canvas.getAttribute("height");
    var context=canvas.getContext("2d");
    //保存状态
    context.save();
    //清空画布
    context.clearRect(0,0,width,height);
    //更新坐标
    this.update(width,height);
    //填充颜色
    context.fillStyle=this.color;
    //移动坐标
    context.translate(this.x,this.y);
    //重新绘制
    context.beginPath();
    context.arc(0,0,this.radius,0,Math.PI*2,true);
    context.fill();
    //恢复状态
    context.restore();
};
```

接下来定义一个动画开始的方法 start()和一个动画暂停的方法 pause()，分别用于“开始”按钮和“暂停”按钮。

start()方法中，使用 setInterval()方法把绘画方法添加到定时器里。在添加定时器之前，先做一个停止动画的操作，防止连续点击开始，造成定时器效果的累加。pause()方法是暂停动画，清除定时器。

动画的开始与暂停方法如示例 8-30 所示。

【示例 8-30】 动画的开始与暂停方法。

```
//暂停动画
animation.pause=function(){
    clearInterval(this.interval);
};
//开始动画
```

```
animation.start =function(){
    //停止动画
    this.pause();
    //开始动画
    this.interval = setInterval("animation.draw()",this.delay);
};
```

最后，将 animation.start()方法应用于“开始”按钮的 onclick 事件；将 animation.pause()方法应用于“暂停”按钮的 onclick 事件。应用如示例 8-26 所示。

此时运行代码，通过操作“开始”按钮和“暂停”按钮，可以看到令人惊奇的动画效果。如果你想让图像变得更加复杂，可以修改绘图方法 animation.draw()。

在 Canvas 里实现动画并不难，总共 100 多行代码，而且不需要借助任何插件，就能实现动画效果。这里实现的只是一个简单的动画，如果你想实现更高级、更复杂的应用，Canvas 也能做到。

8.5　小　　结

本章全面讲解了 Canvas 绘图的各种方法。重点讲解了路径、曲线、图像、渐变、模式和像素操作等绘图方法。其中曲线成形原理涉及较深数学知识，不容易理解；裁剪区域和模式比较抽象，也比较重要，需要多练习、多琢磨；关于像素涉及的数据的底层操作，也是一个难点。

Canvas 绘图是 HTML 5 中一个相当重要的应用，在绘图基础上，不仅可以制作动画，还可以开发游戏。下一节将介绍新型的基于 HTML 5 的多媒体应用：音频和视频。

8.6　习　　题

【习题 1】如何获得二维的绘图上下文对象？请编写相应的代码进行说明。

【习题 2】绘制矩形边框有哪几种方法？请简要说明。

【习题 3】绘制曲线有哪几种方法？请简要说明。

【习题 4】Canvas 绘图中的图形变换是通过改变坐标系统来实现的，请列举常用的变换方法。

【习题 5】编写一段程序，把图片以底片的形式插入到画布中。

第 9 章　便捷的音频和视频

在 HTML 5 之前，在线的音频和视频都是借助 Flash 或第三方工具实现的，现在 HTML 5 也支持这方面的功能。在一个支持 HTML 5 的浏览器中，不需要安装任何插件就能播放音频和视频。原生地支持音频和视频，为 HTML 5 注入了巨大的发展潜力。本章将介绍 HTML 5 中的两个重要元素——audio 和 video，分别用于实现音频和视频，又称为多媒体。对于这两个元素，HTML 5 为开发者提供了标准的、集成的 API。

9.1　audio 和 video 基础知识

HTML 5 对多媒体的支持是顺势发展，只是目前还没有规范得很完整，各种浏览器的支持，差别也很大。如果深入理解 HTML 5 的 audio 和 video 两个元素，有必要对其相关的多媒体技术有一定的了解。

9.1.1　在线多媒体的发展

1. 在线视频的发展

早在 2000 年，在线视频都是借助第三方工具实现的，如 RealPlayer 和 QuickTime 等，但它们存在隐私保护问题或兼容性问题。

HTML 规范的发展与浏览器息息相关，当 Microsoft 赢得了 2001 年的浏览器大战时，即停止了对 IE 浏览器功能的改进。而 W3C 也声明 HTML 规范已经“过时”，转而关注 XHTML 和 XHTML 2，严谨的数据规范和验证，弱化了 HTML 本身的功能。此时没有人认为，在 HTML 中实现视频播放是个好主意。

然而根据实际的需要，开发人员仍然要在网页上实现多媒体功能，进而转向 Flash 的改进功能。

2002 年，Macromedia 为了满足使用 Flash Video 开发人员的需求，引入了 Sorenson Spark。2003 年，该公司使用 VP 6 编解码器（codec）引入了外部视频 FLV 格式。在当时，这是高质量、高压缩的。由此使用 Flash 开发的在线视频，有了近十年的发展，Flash Player 的安装库也变得越来越大，Flash Video 几乎没有缺点，Flash Video 已经发展成为事实上的 Web 标准。

2. 仍然存在的问题

如果在读者的网站上放置一个 Flash 视频，通常需要对 Adobe ActionScript 和专有工具

有很强的理解，才能编码视频和创建播放器控件。一个 Flash 对象的嵌入代码已存在很多年了，如示例 9-1 所示的代码，就算读者研究很长时间，也不能让它简单多少。

在 HTML 5 之前，要在网页中添加音频或视频，最简单、最通用的方法，是使用 Flash。下面我们回顾一下音频或视频的实现方式，如示例 9-1 所示。

【示例 9-1】 Flash 视频嵌入示例。

```
<object id="video" height="400" width="600" codebase="http://download.
macromedia.com/"
       classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" >
   <param value="myVideoPlayer.swf" name="movie" />
   <param value="true" name="allowFullScreen" />
   <param value="all" name="allowNetworking" />
   <param value="always" name="allowScriptAccess" />
   <param value="opaque" name="wmode" />
   <param value="myVideoFile.flv" name="FlashVars" />
   <embed height="520" width="528" src="mds player.swf"
        id=" video" wmode="opaque" allowscriptaccess="always" allownetwor
        king="all"
         allowfullscreen="true" swf="myVideoPlayer.swf"
        flashvars="myVideoFile.flv"
        pluginspage="http://www.macromedia.com/go/getflashplayer"
         type="application/x-shockwave-flash" quality="high" />
</object>
```

示例 9-1 中的代码不兼容 IE 浏览器，如果要兼容 IE 浏览器，需要增加同样多的代码。这种实现方式的缺点是代码较长，最重要的是需要安装 Flash 插件，并非所有浏览器都拥有同样的插件。

3. HTML 5视频简介

在 HTML 5 中，不但不需要安装其他插件，而且实现还很简单。播放一个视频只需要一行代码，如：

```
<video src="resources/test.mp4" autoplay></video>
```

可见，在 HTML 5 中省去了许多不必要的信息。

在 HTML 5 中实现多媒体，不需要知道数据的类型，因为标签已经指明；也不需要设置版本信息，因为不涉及这方面的信息；可以由 CSS 样式表来控制尺寸，因为它们是页面元素。这些原生的优势，是其他任何第三方插件都无法企及的。

9.1.2　多媒体术语

要进行视频开发，读者需要明白多媒体方面的术语。音频和视频都会涉及两个方面的概念：文件格式和编解码器。

1. 视频文件格式

不论是音频还是视频，都只是一个压缩的容器文件。关于视频文件，包含了音频轨道、视频轨道和一些元数据（封面、标题、字幕等）。不同视频格式的视频文件，所属的视频

容器也不一样。目前比较流行的视频格式如下：

- Audio Video Interleaved（.avi）
- Flash Video（.flv）
- MPEG-4（.mp4）
- Matroska（.mkv）
- Ogg（.ogv）

2. 音频和视频编解码器

编解码器是一些算法代码，用来处理视频、音频或者其元数据的编码格式。对音频或视频文件进行编码，可使得文件大大缩小，方便在因特网上传输，因为在网络多媒体发展方面，网络带宽是一个很大的瓶颈。以下是 HTML 5 音频文件格式及其各自的编解码器。

- MP3：使用 ACC 音频。
- Wav：使用 Wav 音频。
- Ogg：使用 OggVorbis 音频。

以下是 HTML 5 视频文件格式及其各自的编解码器。

- MP4：使用 H.264 视频、AAC 音频。
- WebM：使用 VP 8 视频、OggVorbis 音频。
- Ogg：使用 Theora 视频、OggVorbis 音频。

H.264 编解码器被广泛采用，因此读者所使用的大多数编码软件都可以编码一个 MP4 视频。WebM 是新兴的，但是工具都已经可以使用。Ogg 是开源的，但是还没有广泛使用，因此只有少数几个工具可供其使用。

提示： MP4 容器、H.264 视频编解码器及 ACC 音频编解码器都是 MPEG LA Group 专利的专有格式。对于个人网站或者仅有少量视频的公司，这不是问题。然而对于那些有大量视频的公司要非常注意许可证和费用，因为这可能会影响他们盈亏的底线。MP4 容器及其编解码器对终端用户通常是免费的。

WebM 和 Ogg 容器，VP8 和 Theora 视频编解码器及 Vorbis 音频编解码器都是 Berkeley Software Distribution License 授权的、免版费和开放源码的。视频可以无成本制作、分发和观看。然而，传言 VP8 可能侵害一些 H.264 专利，故总是保持最新更新。

9.1.3　HTML 5 多媒体文件格式

在写作本书的时候，很多浏览器已经实现了对 HTML 5 中的 audio 元素和 video 元素的支持。

1. 音频格式的支持情况

目前，audio 元素支持的音频格式是 MP3、Wav 和 Ogg，在各种主流浏览器中的支持情况如表 9.1 所示。

表 9.1　audio元素音频格式的支持情况

浏览器	Wav	MP3	Ogg
IE	/	9.0+	/
Firefox	3.5+	/	3.5+
Safari	3.0+	3.0+	/
Chrome		3.0+	3.0+
Opera	10.5+		10.5+

2. 音频格式的支持情况

目前，video 元素支持的格式是 MP4、WebM 和 Ogg，在各种主流浏览器中的支持情况如表 9.2 所示。

表 9.2　video元素视频格式的支持情况

浏览器	MP4	WebM	Ogg
IE	9.0+	/	/
Firefox	/	4.0+	3.5+
Safari	3.0+	/	/
Chrome	5.0+	6.0+	5.0+
Opera	/	10.6+	10.5+

另外：Mac 上的 Safari 和 Windows 上的 Internet Explorer 9，将支持任何编解码器已经安装在操作系统上的类型。其他浏览器（Firefox、Opera、Chrome）需要具体实现所有视频的编解码器。

9.1.4　功能缺陷及未来趋势

直到现在，仍然不存在完整的音频和视频标准。尽管 HTML 5 提供了音频和视频的规范，但其中所涉及的内容还不够完善。

1. 流式音频和视频

目前的 HTML 5 视频规范中，还没有比特率切换标准，所以对视频的支持仅限于先全部加载完毕再播放的方式。但流式媒体格式是比较理想的格式，在未来的设计中，肯定会在这方面进行规范。

2. 跨源资源的共享

HTML 5 的媒体受到 HTTP 跨源资源共享的限制。HTML 5 针对跨源资源的共享，提供了专门的规范，这种规范不仅仅局限于音频和视频。

3. 全屏控制

从安全角度讲，浏览器中的脚本控制范围不会超出浏览器之外。如果需要控制全屏操作，可能还需要浏览器提供相关的控制功能。

4. 字幕支持

如果在 HTML 5 中对音频或视频进行编程，可能还需要对字幕的控制。基于流行的字幕格式 SRT 的字幕支持规范（WebSRT）仍在编写中，尚未完全纳入规范。

5. 编解码支持

使用 HTML 5 媒体标签的最大缺点在于缺少通用编解码的支持。随着时间的推移，最终会形成一个通用的、高效的编解码器，到时候多媒体的应用形式会比现在更加丰富。未来的发展趋势，一定是我们所期待的那样，或许还会给我们意外的惊喜。

9.2　使用 HTML 5 的 audio 和 video 元素

audio 元素是专门用来在网页中播放网络音频的；video 元素是专门用来在网页中播放视频的。有了这两个元素，就不再需要其他任何插件了，只要浏览器支持 HTML 5 就可以。

HTML 5 为 audio 和 video 元素提供的接口，包含了一系列的属性、方法和事件，这些接口可以帮我们完成针对音频和视频的操作。这两个元素在使用方法和形式上都很类似，下面就一起来介绍这两个元素。

9.2.1　在网页中使用 audio 和 video

1. 在页面中加入音频和视频

这两个元素的使用方法都比较简单，首先以 audio 元素为例，只要指定 audio 标签属性 src 的值为一个音频源文件的路径就可以了，如下所示：

```
<audio src="resources/audio.mp3">
   你的浏览器不支持 audio 元素
</audio>
```

通过这种方法可以把音频数据嵌入到网页上，如果浏览器不支持 HTML 5 的 audio 元素，将会显示替代文字“你的浏览器不支持 audio 元素”。这种不兼容的提示与 Canvas 标签是一样的，也是 HTML 5 处理不兼容的统一方法。

在网页中使用 video 元素添加视频，与 audio 相似，还可以设置 video 元素的尺寸，如下所示：

```
<video src="resources/video.mp4" width="600" height="400">
    你的浏览器不支持 video 元素
</video>
```

通过这种方法即可把视频添加到网页中，浏览器不兼容时，显示替代文字“你的浏览器不支持 video 元素”。对于兼容性的处理方法，也可以增加更加丰富的标签内容，或者增加 Flash 的替代方案。

2. 使用source元素

由于各种浏览器对音频和视频的编解码器的支持不一样，为了能够在各种浏览器中都能正常使用，可以提供多个源文件。这需要使用 source 元素为 audio 元素或 video 元素提供多个备用多媒体文件，如下所示：

```
<audio src="resources/audio.mp3">
  <source src="song.ogg" type="audio/ogg">
  <source src="song.mp3" type="audio/mpeg">
  你的浏览器不支持 audio 元素
</audio>
```

或

```
<video src="resources/video.mp4" width="600" height="400" controls>
  <source src="movie.ogg" type="video/ogg codes='theora,vorbis' ">
  <source src="movie.mp4" type="video/mp4">
  你的浏览器不支持 video 元素
</video>
```

由上面可以看出，我们使用 source 元素替代了 audio 或 video 的标签属性 src。这样，浏览器可以根据自身的播放能力，按照顺序自动选择最佳的源文件进行播放。

此外 source 元素有几个属性：src 属性用于指定媒体文件的 URL 地址；type 属性用于指定媒体文件的类型，属性值为媒体文件的 MIME 类型，该属性值还可以通过 codes 参数指定编码格式。为了提高执行效率，定义详细的 type 属性是必要的。

3. 使用脚本检测浏览器的标签支持情况

也可以使用脚本来判断浏览器是否支持 audio 元素或 video 元素。可以使用脚本动态地创建它，并检测是否存在，脚本如下所示：

```
var support = !!document.createElement("audio").canPlayType;
```

这段脚本会动态创建 audio 元素，然后检查 canPlayType()函数是否存在。通过执行两次逻辑非运算符“!”，将其结果转化成布尔值，就可以确定音频对象是否创建成功。同样，video 元素也可以这样去检查。

9.2.2　audio 和 video 的特性和属性

audio 和 video 的特性（attributes）是用于网页标签的；audio 和 video 的接口属性（properties），用于针对多媒体的编程。下面分别进行介绍。

1. 元素的标签特性（attributes）

❑ src

源文件特性：用于指定媒体文件的 URL 地址。

❑ autoplay

自动播放特性：表示媒体文件加载后自动播放。该属性在标签中的使用方法如下所示：

```
<video src="resources/video.mp4" autoplay></video>
```

❑ controls

控制条特性：表示为视频或音频添加自带的播放控制条。控制条中包括播放/暂停、进度条、进度时间和音量控制等。该属性在标签中的使用方法如下所示：

```
<video src="resources/video.mp4" controls></video>
```

如图 9-1 所示为 Chrome 浏览器自带的控制条。

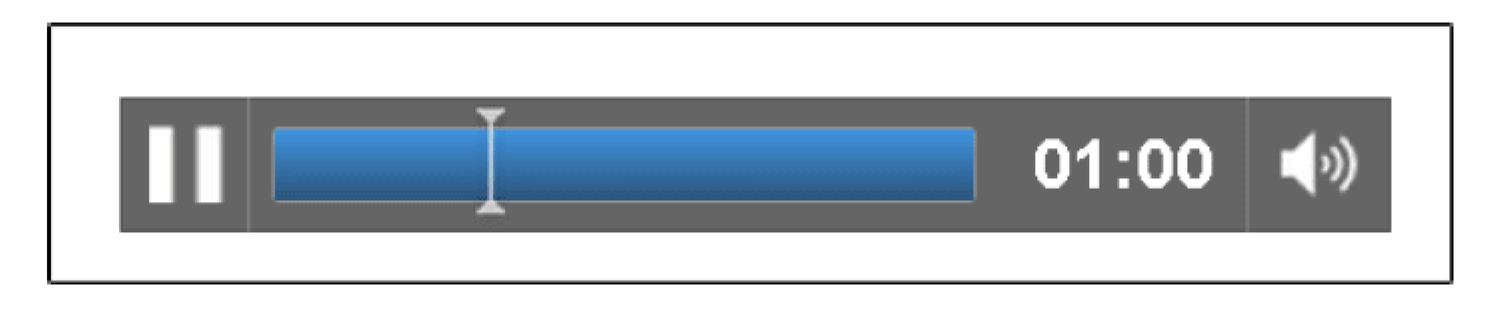


图 9-1　Chrome 自带的播放控制条

❑ loop

循环特性：表示音频或视频循环播放。该属性在标签中的使用方法如下：

```
<video src="resources/video.mp4" controls loop></video>
```

❑ preload

预加载特性：表示页面加载完成后，如何加载视频数据。

该特性有三个值：none 表示不进行预加载；metadata 表示只加载媒体文件的元数据；auto 表示加载全部视频或音频。默认值为 auto。用法如下：

```
<video src="resources/video.mp4" controls preload=”auto”></video>
```

如果设置了 autoplay 属性，则忽略 preload 属性。

❑ poster（video 元素独有的特性）

替代内容属性：用于指定一幅替代图片的 URL 地址，当视频不可用时，会显示该替代图片。用法如下：

```
<video src="resources/video.mp4" controls poster=”images/none.jpg”>
</video>
```

❑ width 和 height（video 元素独有的特性）

宽度和高度特性：用于指定视频的宽度和高度，单位是像素，使用方法如下：

```
<video src="resources/video.mp4" width="600" height="400" controls>
</video>
```

2. 接口属性（properties）

❑ currentSrc（只读）

获取当前正在播放或已加载的媒体文件的 URL 地址。

❑ videoWidth（只读，video 元素特有属性）

获取视频原始的宽度。

❑ videoHeight（只读，video 元素特有属性）

获取视频原始的高度。

- currentTime

获取/设置当前媒体播放位置的时间点，单位为 s（秒）。

- startTime（只读）

获取当前媒体播放的开始时间，通常是 0。

- duration（只读）

获取整个媒体文件的播放时长，单位为 s（秒）。如果无法获取，则返回 NaN。

- volume

获取/设置媒体文件播放时的音量，取值范围在 0.0 到 0.1 之间。

- muted

获取/设置媒体文件播放时是否静音。true 表示静音；false 表示消除静音。

- ended（只读）

如果媒体文件已经播放完毕，则返回 true，否则返回 false。

- played（只读）

获取已播放媒体的 TimesRanges 对象，该对象内容包括已播放部分的开始时间和结束时间。

- paused（只读）

如果媒体文件当前是暂停的或未播放，则返回 true，否则返回 false。

- error（只读）

读取媒体文件的错误代码。正常情况下，error 属性值为 null；有错误时，返回 MediaError 对象 code。

code 有 4 个错误状态值。

  - MEDIA_ERR_ABORTED（值为 1）：中止。媒体资源下载过程中，由于用户操作原因而被中止。
  - MEDIA_ERR_NETWORK（值为 2）：网络中断。媒体资源可用，但下载出现网络错误而中止。
  - MEDIA_ERR_DECODE（值为 3）：解码错误。媒体资源可用，但解码时发生了错误。
  - MEDIA_ERR_SRC_NOT_SUPPORTED（值为 4）：不支持格式。媒体格式不被支持。

- seeking（只读）

获取浏览器是否正在请求媒体数据。true 表示正在请求，false 表示停止请求。

- seekable（只读）

获取媒体资源已请求的 TimesRanges 对象，该对象内容包括已请求部分的开始时间和结束时间。

- networkState（只读）

获取媒体资源的加载状态。该状态有如下 4 个值。

  - NETWORK_EMPTY（值为 0）：加载的初始状态。
  - NETWORK_IDLE（值为 1）：已确定编码格式，但尚未建立网络连接。
  - NETWORK_LOADING（值为 2）：媒体文件加载中。
  - NETWORK_NO_SOURCE（值为 3）：没有支持的编码格式，不加载。

- ❑ buffered（只读）

获取本地缓存的媒体数据的 TimesRanges 对象。TimesRanges 对象可以是个数组。

- ❑ readyState（只读）

获取当前媒体播放的就绪状态。共有如下 5 个值。

  - ➢ HAVE_NOTHING（值为 0）：还没有获取到媒体文件的任何信息。
  - ➢ HAVE_METADATA（值为 1）：已获取到媒体文件的元数据。
  - ➢ HAVE_CURRENT_DATA（值为 2）：已获取到当前播放位置的数据，但没有下一帧的数据。
  - ➢ HAVE_FUTURE_DATA（值为 3）：已获取到当前播放位置的数据，且包含下一帧的数据。
  - ➢ HAVE_ENOUGH_DATA（值为 4）：已获取足够的媒体数据，可正常播放。

- ❑ playbackRate

获取/设置媒体当前的播放速率。

- ❑ defaultPlaybackRate

获取/设置媒体默认的播放速率。

3. 接口属性的使用方法

下面通过一个简单的示例来演示接口属性的使用方法。

【示例 9-2】 视频播放快进。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>视频播放快进</title>
<script type="text/javascript">
function Forward(){
     var el=document.getElementById("myPlayer");
     var time=el.currentTime;       /* 获取属性 currentTime */
     el.currentTime=time+600;       /* 设置属性 currentTime，快进 600s */
}
</script>
</head>
<body>
<video id="myPlayer" src="resources/video.mp4" width="600" height="400"
controls>
</video><br />
<input type="button" value="快进" onClick="Forward()" />
</body>
</html>
```

运行结果如图 9-2 所示。

代码分析：在示例 9-2 中，首先通过脚本获取 video 对象的 currentTime，加上 600 秒（即 10 分钟）后再赋值给对象的 currentTime 属性，即可实现每次快进 10 分钟。由于 currentTime 属性是可读可写的，因此可以给它赋值。

如果接口属性是只读属性，则只能获取该属性的值，不能给该属性赋值。接口属性不能用于 video 标签中，只能通过脚本访问。



图 9-2　视频播放快进

9.2.3　audio 和 video 的方法

HTML 5 为 audio 和 video 元素提供了同样的接口方法，下面一起介绍。

1. 接口方法

❑ load()

加载媒体文件，为播放做准备。通常用于播放前的预加载；还会用于重新加载媒体文件。

❑ play()

播放媒体文件。如果视频没有加载，则加载并播放；如果是暂停的，则变为播放，自动把 paused 属性变为 false。

❑ pause()

暂停播放媒体文件。自动把 paused 属性变为 true。

❑ canPlayType()

测试浏览器是否支持指定的媒体类型。该方法的语法如下：

```
canPlayType(<type>)
```

<type>：指定的媒体类型，与 source 元素的 type 参数的指定方法相同。指定方式如：

“video/mp4”，指定为媒体文件的 MIME 类型，该属性值还可以通过 codes 参数指定编码格式。

该方法可有如下 3 个返回值。

- 空字符串：表示浏览器不支持指定的媒体类型。
- maybe：表示浏览器可能支持指定的媒体类型。
- probably：表示浏览器确定支持指定的媒体类型。

2. 接口方法的使用

下面通过一个简单的示例来演示接口方法的使用。

【示例 9-3】 播放与暂停。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>播放与暂停</title>
<script type="text/javascript">
var videoEl=null;
function Play(){
     videoEl.play();      /* 播放视频 */
}
function Pause(){
     videoEl.pause();   /* 暂停播放 */
}
window.onload=function(){
     videoEl=document.getElementById("myPlayer");
}
</script>
</head>
<body>
<video id="myPlayer" width="600">
  <source src="resources/video.mp4" type="video/mp4">
  你的浏览器不支持 video 元素
</video><br>
<input type="button" value="播放" onclick="Play()" />
<input type="button" value="暂停" onclick="Pause()" />
</body>
</html>
```

运行结果如图 9-3 所示。

代码分析：在示例 9-3 中，设置了两个按钮，分别控制视频的播放与暂停。“播放”按钮通过定义的 Play()函数执行视频的接口方法 play()；“暂停”按钮通过定义的 Pause()函数执行视频的接口方法 pause()。如图 9-3 所示，“播放”和“暂停”按钮均可使用。

有了这些接口方法和接口属性，如果你不喜欢系统自带的控制条，大可以自己做一个好看的。

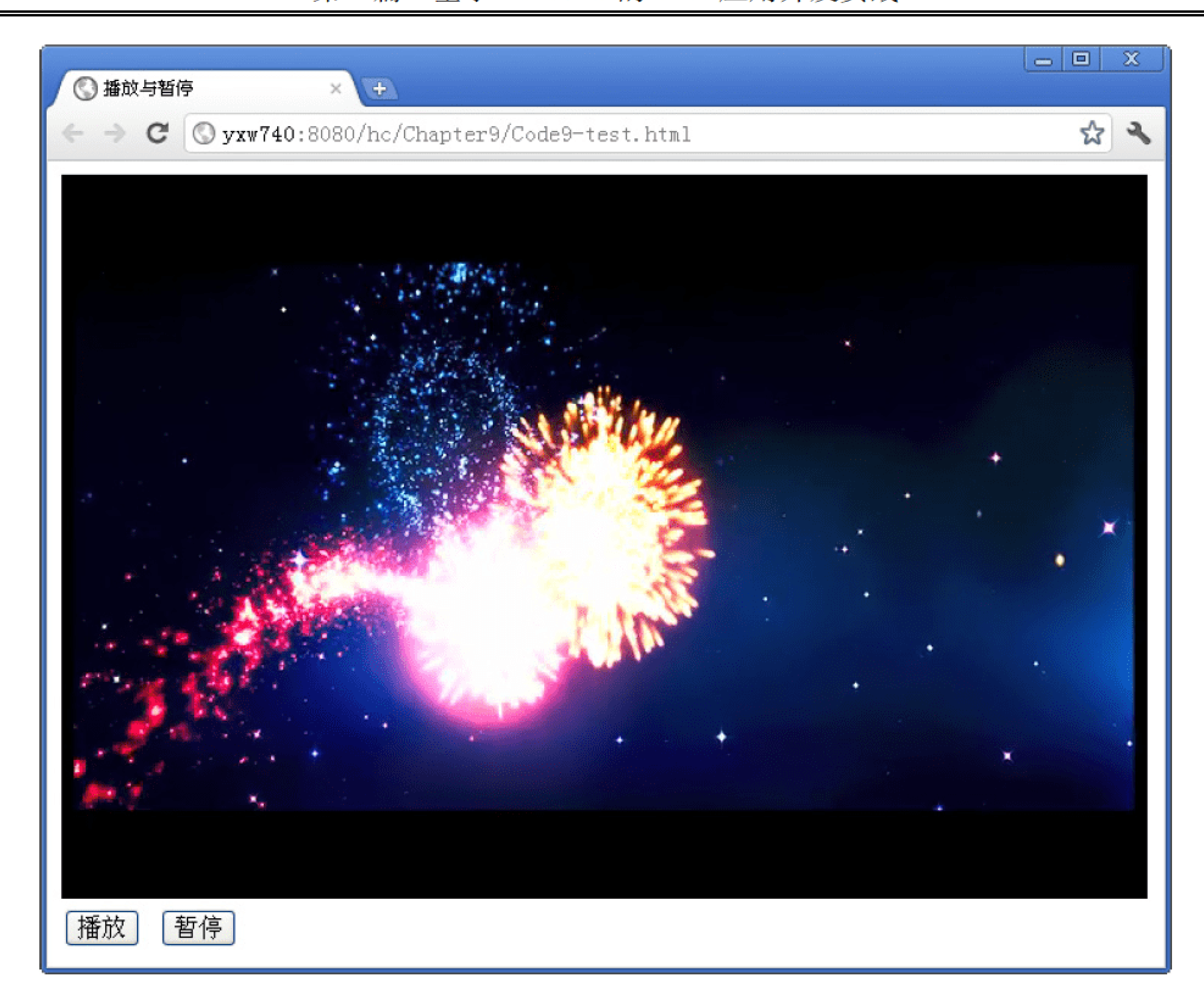


图 9-3　播放与暂停

9.2.4　audio 和 video 的事件

HTML 5 还为 audio 和 video 元素提供了一系列的接口事件。在使用 audio 和 video 元素读取或播放媒体文件的时候，会触发一系列的事件，可以用 JavaScript 脚本来捕获这些事件，并进行相应的处理。

1. 捕获事件的方式

捕获事件有两种方法：一种是添加事件句柄；一种是监听。

在页面的 audio 或 video 标签中添加事件句柄，如下所示：

```
<video id="myPlayer" src="resources/video.mp4" width="600" onplay="video_
playing()"> </video>
```

然后就可以在函数 video_playing()中，添加需要的代码。监听方式：

```
var videoEl=document.getElementById("myPlayer");
videoEl.addEventListener("play",video_playing);    /* 添加监听事件 */
```

2. 接口事件

- play

当执行方法 play()时触发。

- playing

正在播放时触发。

- pause

当执行了方法 pause()时触发。

- timeupdate

当播放位置被改变时触发。可能是播放过程中的自然改变，也可能是人为改变。

- ended

当播放结束后停止播放时触发。

- waiting

在等待加载下一帧时触发。

- ratechange

在当前播放速率改变时触发。

- volumechange

在音量改变时触发。

- canplay

以当前播放速率，需要缓冲时触发。

- canplaythrough

以当前播放速率，不需要缓冲时触发。

- durationchange

当播放时长改变时触发。

- loadstart

当浏览器开始在网上寻找数据时触发。

- progress

当浏览器正在获取媒体文件时触发。

- suspend

当浏览器暂停获取媒体文件，且文件获取并没有正常结束时触发。

- abort

当中止获取媒体数据时触发。但这种中止不是由错误引起的。

- error

当获取媒体过程中出错时触发。

- emptied

当所在网络变为初始化状态时触发。

- stalled

浏览器尝试获取媒体数据失败时触发。

- loadedmetadata

在加载完媒体元数据时触发。

- loadeddata

在加载完当前位置的媒体播放数据时触发。

- seeking

浏览器正在请求数据时触发。

- seeked

浏览器停止请求数据时触发。

9.3　实验室：自定义播放工具条

audio 和 video 元素都有一个默认的播放工具条，如果标签设置 controls 属性，该播放工具条就会显示出来，使用起来也非常方便。但对于设计者来说，千篇一律的风格，总会让人厌倦。本节我们就自己定义一个播放工具条，可以按照自己的风格定义。

1. 案例简介

本节介绍的案例，是定义一个播放工具条，主要功能包括：播放/暂停、快进、慢进、前进、后退、静音、音量等控制。其中播放和暂停是同一按钮，会根据播放状态变换；快进和慢进会改变播放速度；工具条右边是显示关于速率和时间进度的信息；上面还有一个进度条，以更加友好地显示进度，案例效果图如图 9-4 所示。



图 9-4　自定义播放工具条

2. 网页基本元素

【示例 9-4】　自定义播放工具条。

```
<!DOCTYPEHTML>
<html>
<head>
<meta charset="utf-8">
<title>自定义播放工具条</title>
<style type="text/css">
  … / * 样式表省略 */
</style>
</head>
<body>
<video id="myPlayer" src="resources/video.mp4">你的浏览器不支持 video 元素
</video>
<!-- 播放工具条 -->
<div id="controls">
  <div id="bar">       <!-- 进度条 -->
    <div id="progresss"></div>
  </div>
  <div id="slow" class="but" onclick="Slow()">7</div> <!-- 慢进 -->
  <div id="play" class="but" onclick="Play(this)">4;</div><!-- 播放/暂停
-->
  <div id="fast" class="but" onclick="Fast()">8</div><!-- 快进 -->
  <div id="prev" class="but" onclick="Prev()">9</div><!-- 后退 -->
  <div id="next" class="but" onclick="Next()">:</div><!-- 前进 -->
  <div id="muted" onclick="Muted(this)">X</div><!-- 静音控制 -->
  <div class="volume">
<!-- 音量控制 -->
    <input  id="volume"  type="range"  min="0"  max="1"  step="0.1"
    onchange="Volume(this)" />
  </div>
  <div class="info">
      <span id="rate">1</span>fps <span id="info"></span>  <!-- 速率和时
      间进度的信息 -->
  </div>
</div>
</body>
</html>
```

此时，有了样式表的控制，界面外观已经有如图 9-4 所示的界面效果，但还没有播放控制功能。下面我们根据案例的要求，逐步添加脚本以完善播放工具条的功能。

3. 定义全局的视频对象

为了方便调用视频对象，我们把视频对象定义为全局变量。如以下代码所示：

```
<script type="text/javascript">
/* 定义全局视频对象 */
var videoEl=null;
/* 网页加载完毕后，读取视频对象 */
window.addEventListener("load", function(){
    videoEl=document.getElementById("myPlayer")
});
</script>
```

4. 添加播放/暂停、前进和后退功能

播放和暂停使用同一个按钮。暂停时，播放功能有效，可点击播放视频；播放时，暂停功能有效，可点击暂停播放。前进和后退，仅仅是改变了 currentTime 属性的值。

```
<script type="text/javascript">
/* 播放/暂停 */
function Play(e){
    if(videoEl.paused){
          videoEl.play();                              / * 如果暂停，则播放 */
          document.getElementById("play").innerHTML=";"
                                                       /* 显示暂停的文字图像 */
    }else{
        videoEl.pause();                           / * 如果不是暂停，则暂停 */
        document.getElementById("play").innerHTML="4"/* 显示播放的文字图像 */
    }
}
/* 后退：后退一分钟 */
function Prev(){
      videoEl.currentTime-=60;/ * currentTime 属性减去 60s，即向后倒退一分钟 */
}
/* 前进：前进一分钟 */
function Next(){
     videoEl.currentTime+=60;  / * currentTime 属性增加 60s,即向前前进一分钟 */
}
</script>
```

5. 添加慢进和快进功能

慢进和快进是通过改变速率来实现的。默认速率为 1。当速率小于 1 时，每次改变 0.2 的速率；当速率大于 1 时，每次改变的速率为 1。速率改变后，会在播放工具条中显示出来。

```
<script type="text/javascript">
/* 慢进：小于等于 1 时，每次只减慢 0.2 的速率；大于 1 时，每次减 1 */
function Slow(){
    if(videoEl.playbackRate<=1)
        videoEl.playbackRate-=0.2;
    else{
        videoEl.playbackRate-=1;
    }
    document.getElementById("rate").innerHTML=fps2fps(videoEl.playbackRa
    te);  /* 显示播放速率 */
}
/* 快进：小于 1 时，每次只加快 0.2 的速率；大于 1 时，每次加 1*/
function Fast(){
    if(videoEl.playbackRate<1)
        videoEl.playbackRate+=0.2;
    else{
        videoEl.playbackRate+=1;
    }
    document.getElementById("rate").innerHTML=fps2fps(videoEl.playbackRa
    te);  /* 显示播放速率 */
}
```

```
/* 速率数值处理 */
function fps2fps(fps){
    if(fps<1)
        return fps.toFixed(1);
    else
        return fps
}
</script>
```

6. 添加静音和音量的功能

静音时，音量为 0；不静音时，音量还原为视频的音量。音量是通过拖动滑块来调整的，当拖动滑块时，即触发事件，修改视频的音量。

```
<script type="text/javascript">
/* 静音 */
function Muted(e){
    if(videoEl.muted){
        videoEl.muted=false;           /* 消除静音 */
        e.innerHTML="X";               /* 显示声音的文字图标 */
        document.getElementById("volume").value=videoEl.volume;
                                       /* 还原音量 */
    }else{
        videoEl.muted=true;            /* 静音 */
        e.innerHTML="x";               /* 显示静音的文字图标 */
        document.getElementById("volume").value=0;  /* 音量修改为 0 */
    }
}
/* 调整音量 */
function Volume(e){
    videoEl.volume=e.value;  /* 修改音量的值 */
}
</script>
```

7. 添加进度显示功能

网页加载完成后，执行进度处理函数，并把进度处理函数添加至视频对象的 timeupdate 事件中，以便能实时更新进度信息。

```
<script type="text/javascript">
/* 进度信息：控制进度条，并显示进度时间 */
function Progresss(){
     var el=document.getElementById("progresss");
     el.style.width = (videoEl.currentTime/videoEl.duration)*720 +"px"
     document.getElementById("info").innerHTML=s2time(videoEl.current
     Time)+"/"+s2time(videoEl.duration);
}
/* 把秒处理为时间格式 */
function s2time(s){
    var m=parseFloat(s/60).toFixed(0);
    s=parseFloat(s%60).toFixed(0);
    return (m<10?"0"+m:m) +":"+ (s<10?"0"+s:s);
}
/* 网页加载完毕后，把进度处理函数添加至视频对象的 timeupdate 事件中 */
window.addEventListener("load",
```

```
function(){videoEl.addEventListener("timeupdate",Progresss)});
/* 给 window.onload 事件添加进度处理函数 */
window.addEventListener("load",Progresss);
</script>
```

至此，播放工具条的功能已经完成。此时，点击播放工具条中的按钮，都能操作播放中的视频。

9.4　小　　结

本章主要讲解了 HTML 5 的音频和视频在互联网方面的发展轨迹，以及接口中的属性、方法和事件。重点讲解了音频和视频的基础知识、HTML 5 提供的音频和视频的接口及如何使用这些接口。在实验的案例中，演示了主要的接口属性、方法和事件。比较难的是对媒体文件中的元数据概念的理解；音频和视频都包含两类属性，容易混淆，这也是本章的难点。下一章，我们将介绍 HTML 5 在表单方面的改进。

9.5　习　　题

【习题 1】如何在网页中使用音频和视频？
【习题 2】如何在网页中加入循环播放的视频？
【习题 3】HTML 5 为音频和视频提供了哪些可用的方法？

第 10 章　不可思议的表单

一直以来，表单都是 Web 的核心技术之一，多数在线应用都是通过表单来交互完成的。作为交互的数据载体，有必要为用户提供更加友好的操作和严谨的表单验证，这对于大多数 Web 开发人员来说，是一直在做的事情。如今，HTML 5 正在努力地简化我们的工作。为此，HTML 5 不但增加了一系列功能性的表单、表单元素、表单特性，还增加了自动验证表单的功能。正如您所期待的那样，一切将变得不可思议。本章将深入探讨 HTML 5 的表单技术。

10.1　HTML 5 表单概述

HTML 5 对表单的发展，是适应互联网发展的需要，也是在适应开发者的需要。相比以前，HTML 5 的表单功能有了革命性的改变。

10.1.1　HTML 表单的进化

1．早期的发展

早在 20 世纪 90 年代的 HTML 4 规范中，表单功能已经发展得非常完善。HTML 4 的表单很单纯，支持最基本的数据输入，所有的表单应用都可以使用，时至今日，我们仍然在使用它。

随着 Web 应用的发展，表单功能太过简单，在处理复杂业务的过程中，显得能力有限，而且还受到网络设备的限制。基于这个原因，出现了基于 XML 的 XHTML 规范，于此同时出现了 Xform 表单，基于 HTML 4 的表单也停止了发展。

XForms 试图突破当前 HTML Forms 模型的一些限制，而且 XForms 的最大特色是包含了客户端验证的功能，避免使用大量的 JavaScript 脚本验证。在当时，XForm 被称为“下一代 Web 表单”。

由于 XForms 是基于 XML 的，在一定程度上弱化了标签本身的功能；由于其比较灵活，表单也跟着复杂了。这在实际的使用过程中，并没有得到广泛的发展。

2．HTML 5表单的出世

在实际的表单应用中，一些特殊的数据输入需要一个独立的规则，如邮件、网址等，我们都会提供一个特定的格式限定和验证。

由于移动互联网的快速发展，在面向移动设备的时候，通过识别表单类型，可以提供

更友好的用户体验，如可以呈现不同的屏幕键盘等。

HTML 5 的表单，在原有表单的基础上，参照 Xform 的一些验证功能，再结合实际发展的需要，制定了新型的功能性表单，并且支持表单验证。

在做表单处理的时候，最常用的就是表单验证了。一般的验证会写很多冗长的 JavaScript 代码，或者借助一些基于 JavaScript 的验证框架，如目前比较流行的 jQuery 的验证框架。HTML 5 发展了这些表单，把具有特定规则意义的表单，扩展一些特有的特性，作为表单的原始功能；验证表单的功能，也作为表单本身应具备的功能，原生地被支持。

HTML 5 的表单，无论是在表现方面还是在功能方面都非常优越，开发起来可以不用那么复杂。HTML 5 的表单的目的就是让这一切友好的应用变得简单。

10.1.2　当前的支持情况

由于 HTML 5 的规范还在渐进发展中，各个浏览器的支持程度也不一样，因此在使用 HTML 5 表单功能时，应尽量避免滥用，最好再提供替代解决方案。

根据 HTML 5 的设计原则，在旧的浏览器中，新的表单空间会平滑降级，不需要判断浏览器的支持情况。

虽然 HTML 5 表单的一些规范还没有获得浏览器的支持，但仍然可以借鉴表单规范的设计思想为我所用，如果浏览器不支持，我们就通过其他方式帮助实现。

10.2　新增表单输入类型

HTML 5 大幅度地改进了 input 元素的类型。不同类型的表单所附加的功能也不相同。到目前为止，支持最多、最全面的是 Opera 10 浏览器。对于不支持新增表单类型的浏览器来说，会默认识别为 text 类型。

10.2.1　新增的表单输入类型

这些新增的表单类型不但方便进行表单验证，而且在用户体验方面留下了极大的提升空间。下面针对这些 input 元素的类型逐步讲解。

1. url类型

url 类型的 input 元素，是专门为输入 url 地址定义的文本框。在验证输入文本的格式时，如果该文本框中的内容不符合 url 地址的格式，会提示验证错误。该表单类型的使用方法如下：

```
<input type="url" name="webUrl" id="webUrl" value="http://www.baidu.com"
/>
```

2. email类型

email 类型的 input 元素，是专门为输入 E-mail 地址定义的文本框。在验证输入文本的

格式时，如果该文本框中的内容不符合 E-mail 地址的格式时，会提示验证错误。该表单类型的使用方法如下：

```
<input type="email" name="myEmail" id="myEmail" value="yxw740@163.com" />
```

此外 email 类型的 input 元素还有一个 multiple 属性，表示在该文本框中可输入用逗号隔开的多个邮件地址。

3．range类型

range 类型的 input 元素把输入框显示为滑动条，为某一特定范围内的数值选择器。它还具有 min 和 max 特性，表示选择范围的最小值（默认为 0）和最大值（默认为 100）；还有 step 特性，表示拖动步长（默认为 1）。该表单类型的显示效果如图 10-1 所示。该表单类型的使用方法如下：

```
<input type="range" name="volume" id="volume" min="0" max="1" step="0.2"
/>
```

4．number类型

number 类型的 input 元素是专门为输入特定的数字而定义的文本框。与 range 类似，都具有 min、max 和 step 特性，表示允许范围的最小值、最大值和调整步长。该表单类型的显示效果如图 10-2 所示，该表单类型的使用方法如下：

```
<input type="number" name="score" id="score" min="0" max="10" step="0.5"
/>
```

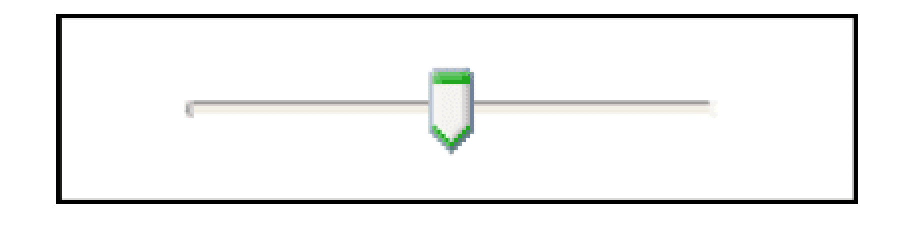


图 10-1　range 类型的外观

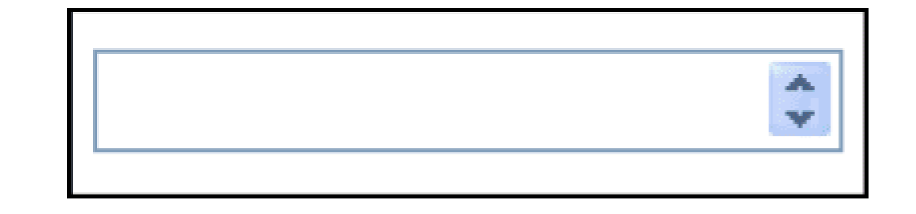


图 10-2　number 类型的外观

5．tel类型

tel 类型的 input 元素是专门为输入电话号码而定义的文本框，没有特殊的验证规则。

6．search类型

search 类型的 input 元素是专门为输入搜索引擎关键词定义的文本框，没有特殊的验证规则。

7．color类型

color 类型的 input 元素，默认会提供一个颜色选择器，主流浏览器还没有支持它。

8．date类型

date 类型的 input 元素是专门用于输入日期的文本框，默认为带日期选择器的输入框，目前仅 Opera 浏览器支持它。

9．month、week、time、datetime、datetime-local类型

month、week、time、datetime、datetime-local 类型的 input 元素与 date 类型的 input 元素类似，都会提供一个相应的选择器。其中，month 会提供一个月选择器；week 会提供一个周选择器；time 会提供时间选择器；datetime 会提供完整的日期和时间（包含时区）的选择器；datetime-local 也会提供完整的日期和时间（不包括时区）选择器。

10.2.2　面向未来的新型表单

就目前而言，使用 HTML 5 新增的表单输入类型，表现出来的好处是非常有限的，还要处理向下的兼容性，从某种角度讲，反而增加了工作量。

或许有人会问，定义这么多的表单类型是多此一举。因为在表单验证方面，对于很多类型的表单，使用传统的方法也能轻松完成。

HTML 5 的表单设计的功能主要面向两方面：一方面是用于表单验证，但不是每个新增的表单类型，都有非常具体的验证规则；另一方面是用于未来移动设备中，通过识别表单类型来提供更加具体而友好的用户体验，才是新增表单类型的主要目的。

例如，在 iPhone 上，当输入邮件时，会提供一个带有@符号的屏幕键盘，不符合邮件规范的特殊符号则不显示。

在未来的移动设备上，键盘将在屏幕内显示，鼠标也将消失。这方面已经在苹果设备上充分体现，代表了未来的方向。HTML 5 顺应了移动互联网的发展趋势，在移动设备的环境下，新增一系列表单类型就显得非常有必要了。

HTML 5 在走向移动互联网的过程中，对表单在这方面的改进，是一种前瞻性的设计，是面向未来的设计。

10.3　新增表单特性及元素

如果开发一个用户体验非常好的页面，潜在地需要写大量的代码，而且还需要考虑兼容性问题。使用 HTML 5 表单的某些特性，可以开发出前所未有的页面效果，可以写更少的代码，并能解决传统开发中碰到的一些问题。

1．form特性

通常，从属于表单的元素必须放在表单内部。但是在 HTML 5 中，可以把从属于表单的元素放在任何地方，然后指定该元素的 form 特性值为表单的 id，这样该元素就从属于表单了。form 特性的使用方法如示例 10-1 所示。

【示例 10-1】 form 特性的使用方法。

```
<input name="name" type="text" form="form1" required />
<form id="form1">
  <input type="submit" value="提交" />
</form>
```

代码分析：在示例 10-1 中，可输入的 input 元素在表单 form 之外，由于 input 元素的 form 特性值指定了表单的 id，说明该元素从属于表单。当点击提交按钮时，会验证该从属元素。目前，form 特性已获得多个主流浏览器的支持。

2．formaction特性

每个表单都会通过 action 特性把表单内容提交到另外一个页面。在 HTML 5 中，为不同的提交按钮分别添加 formaction 特性，该特性会覆盖表单的 action 特性，将表单提交至不同的页面。formaction 特性的使用方法如示例 10-2 所示。

【示例 10-2】 formaction 特性的使用方法。

```
<form id="form1" method="post">
  <input name="name" type="text" form="form1" />
  <input type="submit" value="提交到 Page1" formaction="?page=1" />
  <input type="submit" value="提交到 Page2" formaction="?page=2" />
  <input type="submit" value="提交到 Page3" formaction="?page=3" />
  <input type="submit" value="提交" />
</form>
```

代码分析：在示例 10-2 中，添加了 4 个提交按钮，其中前 3 个提交按钮设置了 formaction 特性，提交表单时，会优先使用 formaction 特性值作为表单提交的目标页面。

目前，formaction 特性已获得多个主流浏览器的支持。

3．formmethod、formenctype、formnovalidate、formtarget特性

这 4 个特性的使用方法与 formaction 特性一致，设置在提交按钮上，可以覆盖表单的相关特性。formmethod 特性可覆盖表单的 method 特性；formenctype 特性可覆盖表单的 enctype 特性；formnovalidate 特性可覆盖表单的 novalidate 特性；formtarget 特性可覆盖表单的 target 特性。

4．placeholder特性

当用户还没有把焦点定位到输入文本框的时候，可以使用 placeholder 特性向用户提示描述性的信息，当该输入文本框获取焦点时，该提示信息就会消失。该特性的使用方法如下：

```
<input name="name" type="text" placeholder="请输入关键词" />
```

显示效果如图 10-3 所示。

placeholder 特性还可用于其他输入类型的 input 元素，如 url、email、number、search、tel 和 password 等。目前，placeholder 特性已获得多个主流浏览器的支持。

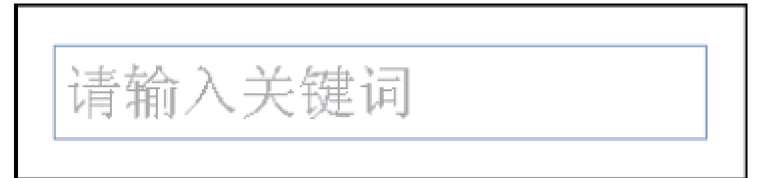


图 10-3　使用 placeholder 特性

5．autofocus特性

使用 autofocus 特性可用于所有类型的 input 元素，当页面加载完成时，可自动获取焦点。每个页面只允许出现一个有 autofocus 特性的 input 元素。如果为多个 input 元素设置了 autofocus 特性，则相当于未指定该行为。该特性的使用方法如下：

```
<input name="key" type="text" autofocus />
```

自动获取焦点的功能也要防止滥用。如果页面加载缓慢，用户又做了一部分操作，这时如果焦点发生莫名其妙的转移，用户体验是极不友好的。目前，autofocus 特性已获得多个主流浏览器的支持。

6. autocomplete特性

IE 的早期版本就已经支持 autocomplete 特性。autocomplete 特性可应用于 form 元素和输入型的 input 元素，用于表单的自动完成。autocomplete 特性会把输入的历史记录下来，当再次输入的时候，会把输入的历史记录显示在一个下拉列表里，以实现自动完成输入。该特性的使用方法如下：

```
<input name="key" type="text" autocomplete="on" />
```

autocomplete 特性有 3 个值，可以指定“on”、“off”和“”（不指定）。不指定值时，使用浏览器的默认设置。由于不同的浏览器默认值不相同，因此当需要使用自动完成的功能时，最好显式地指定该特性。目前，autofocus 特性已获得所有主流浏览器的支持。

7. list特性和datalist元素

通过组合使用 list 特性和 datalist 元素，可以为某个可输入的 input 元素定义一个选值列表。使用 datalist 元素构造选值列表；设置 input 元素的 list 特性值为 datalist 元素的 id 值，可实现二者的绑定。其使用方法如示例 10-3 所示。

【示例 10-3】 list 特性和 datalist 元素的使用方法。

```
<input name="email" type="email" list="emaillist" />
<datalist id="emaillist">
  <option value="test1@test.com">test1@test.com</option>
  <option value="test2@test.com">test2@test.com</option>
</datalist>
```

运行结果如图 10-4 所示。

代码分析：在示例 10-3 中，使用 datalist 元素构造了一个选值列表，id 为 emaillist；input 元素通过把 list 特性值设为 emaillist，绑定了该选值列表，运行结果如图 10-4 所示。

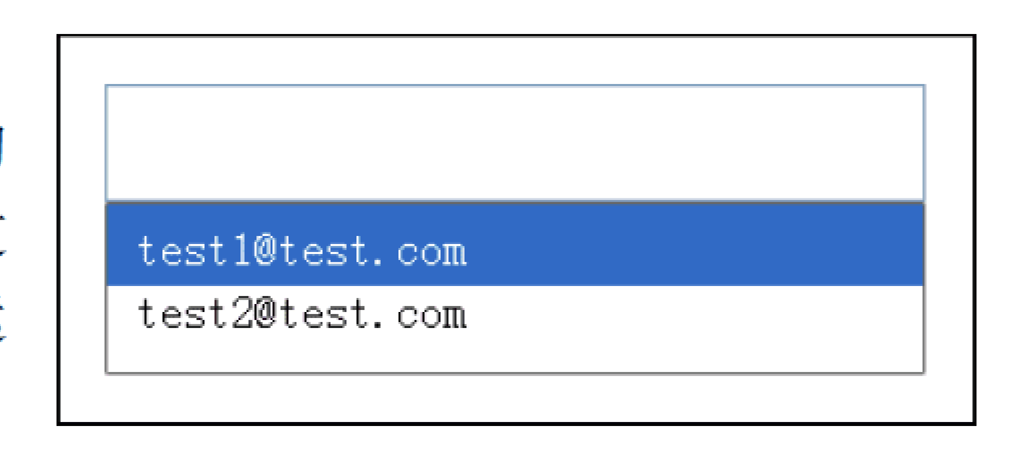


图 10-4　list 和 datalist 使用示例

8. keygen元素

keygen 元素提供了一种安全的方式来验证用户。该元素有密钥生成的功能，当提交表单时，会分别生成一个私人密钥和一个公共密钥。其中私人密钥保存在客户端，公共密钥则通过网络传输至服务器。这种非对称加密的方式，为网页的数据安全提供了更大的保障。该元素的使用方法如示例 10-4 所示。

【示例 10-4】 keygen 元素的使用方法。

```
<form action="">
  <input type="text" name="name" /><br>
  Encryption:
  <keygen name="security" />   <!-- 加入密钥安全 -->
```

```
  <br><input type="submit" />
</form>
```

keygen 元素提供了中级和高级的加密算法，显示的是一个类似 select 元素的下拉框，可以选择加密等级。目前，keygen 元素已获得多数主流浏览器的支持。

9. output元素

output 元素用于不同类型的输出，如用于计算结果或脚本的输出等。output 元素必须从属于某个表单，即写在表单的内部。该元素的使用方法如下：

【示例 10-5】 output 元素的使用方法。

```
<form oninput="x.value=volume.value">
  <input type="range" name="volume" value="50" />
  <output name="x"></output>
</form>
```

由于 range 类型的 input 元素表现为一个滑块，不显示数值，因此这时使用 output 元素协助显示其值。目前，output 元素已获得多数主流浏览器的支持。

10.4　表单验证 API

HTML 5 为表单验证提供了极大的方便，在验证表单的方式上显得更加灵活。表单验证，首先会基于前面讲解的表单类型的规则进行验证；其次是为表单元素提供了一些用于辅助表单验证的特性；更重要的是，HTML 5 还提供了专门用于表单验证的属性、方法和事件。

10.4.1　与验证有关的表单元素特性

HTML 5 提供了用于辅助表单验证的元素特性。利用这些特性，可以为后续的表单自动验证提供验证依据。下面就对这些新的特性进行讲解。

1. required特性

一旦为某个表单内部的元素设置了 required 特性，那么此项的值不能为空，否则无法提交表单。以文本输入框为例，只需要添加 required 特性即可，使用方法如下：

```
<input name="name" type="text" placeholder="Full Name" required />
```

如果该项为空，则无法提交。required 特性可用于大多数输入或选择元素，隐藏的元素除外。

2. pattern特性

pattern 特性用于为 input 元素定义一个验证模式。该特性值是一个正则表达式，提交时，会检查输入的内容是否符合给定的格式，如果输入内容不符合格式，则不能提交。使用方法如下：

```
<input name="code" type="text" value="" pattern="[0-9]{6}" placeholder="6
位邮政编码" />
```

使用 pattern 特性验证表单非常灵活。例如，前面讲到的 email 类型的 input 元素，使用 pattern 特性完全可以实现相同的验证功能（不过 email 类型的用途却不仅限于此）。

3. min、max和step特性

min、max 和 step 特性专门用于指定针对数字或日期限制。min 特性表示允许的最小值；max 特性表示允许的最大值；step 特性表示合法数据的间隔步长。使用方法如下：

```
<input type="range" name="volume" id="volume" min="0" max="1" step="0.2"
/>
```

该示例中，最小值是 0，最大值是 1，步长为 0.2，合法的取值有 0、0.2、0.4、0.6、0.8、1。

4. novalidate特性

novalidate 特性用于指定表单或表单内的元素在提交时不验证。如果 form 元素应用 novalidate 特性，则表单中的所有元素在提交时都不再验证。使用方法如下：

```
<form action="demo_form.asp" novalidate="novalidate">
  <input type="email" name="user email" />
  <input type="submit" />
</form>
```

则提交时，不会验证表单。

10.4.2　表单验证的属性

表单验证的属性均是只读属性，用于获取表单验证的信息。

1. validity 属性

该属性获取表单元素的 ValidityState 对象，该对象包含 8 个方面的验证结果。ValidityState 对象会持续存在，每次获取 validity 属性时，返回的是同一个 ValidityState 对象。以一个 id 特性为“username”的表单元素为例，validity 属性的使用方法如下：

```
var validityState=document.getElementById("username").validity;
```

关于 ValidityState 对象将在下一节讲解。

2. willValidate属性

该属性获取一个布尔值，表示表单元素是否需要验证。如果表单元素设置了 required 特性或 pattern 特性，则 willValidate 属性的值为 true，即表单的验证将会执行。仍然以一个 id 特性为“username”的表单元素为例，willValidate 属性的使用方法如下：

```
var willValidate=document.getElementById("username").willValidate;
```

3．validationMessage属性

该属性获取当前表单元素的错误提示信息。一般设置 reuired 特性的表单元素，其 validationMessage 属性值一般为“请填写此字段”。仍然以一个 id 特性为“username”的表单元素为例，validationMessage 属性的使用方法如下：

```
var validationMessage=document.getElementById("username").validation
Message;
```

此属性为只读属性，不能直接更改。不过可以使用 setCustomValidity()方法（后面介绍）来改变该值。

10.4.3　ValidityState 对象

ValidityState 对象是通过 validity 属性获取的，该对象有 8 个属性，分别针对 8 个方面的错误验证，属性值均为布尔值。

1．valueMissing属性

必填的表单元素的值为空。

如果表单元素设置了 required 特性，则为必填项。如果必填项的值为空，就无法通过表单验证，valueMissing 属性会返回 true，否则返回 false。

2．typeMismatch属性

输入值与 type 类型不匹配。

HTML 5 新增的表单类型如 email、number、url 等，都包含一个原始的类型验证。如果用户输入的内容与表单类型不符合，则 typeMismatch 属性将返回 true，否则返回 false。

3．patternMismatch属性

输入值与 pattern 特性的正则不匹配。

表单元素可通过 pattern 特性设置正则表达式的验证模式。如果输入的内容不符合验证模式的规则，则 patternMismatch 属性将返回 true，否则返回 false。

4．tooLong属性

输入的内容超过了表单元素的 maxLength 特性限定的字符长度。

表单元素可使用 maxLength 特性设置输入内容的最大长度。虽然在输入的时候会限制表单内容的长度，但在某种情况下，如通过程序设置，还是会超出最大长度限制。如果输入的内容超过了最大长度限制，则 tooLong 属性返回 true，否则返回 false。

5．rangeUnderflow属性

输入的值小于 min 特性的值。

一般用于填写数值的表单元素，都可能会使用 min 特性设置数值范围的最小值。如果输入的数值小于最小值，则 rangeUnderflow 属性返回 true，否则返回 false。

6．rangeOverflow属性

输入的值大于 max 特性的值。

一般用于填写数值的表单元素，也可能会使用 max 特性设置数值范围的最大值。如果输入的数值大于最大值，则 rangeOverflow 属性返回 true，否则返回 false。

7．stepMismatch属性

输入的值不符合 step 特性所推算出的规则。

用于填写数值的表单元素，可能需要同时设置 min、max 和 step 特性，这就限制了输入的值必须是最小值与 step 特性值的倍数之和。如范围从 0 到 10，step 特性值为 2，因为合法值为该范围内的偶数，其他数值均无法通过验证。如果输入值不符合要求，则 stepMismatch 属性返回 true，否则返回 false。

8．customError属性

使用自定义的验证错误提示信息。

有时候，不太适合使用浏览器内置的验证错误提示信息，需要自己定义。当输入值不符合语义规则时，会提示自定义的错误提示信息。

通常使用 setCustomValidity()方法自定义错误提示信息：setCustomValidity(message)会把错误提示信息自定义为 message，此时 customError 属性值为 true；setCustomValidity(“”)会清除自定义的错误信息，此时 customError 属性值为 false。

10.4.4　表单验证的方法

HTML 5 为我们提供了两个用于表单验证的方法。

1．checkValidity()方法

显式验证方法。每个表单元素都可以调用 checkValidity()方法（包括 form），它返回一个布尔值，表示是否通过验证。默认情况下，表单的验证发生在表单提交时，如果使用 checkValidity()方法，可以在需要的任何地方验证表单。一旦表单元素没有通过验证，则会触发 invalid 事件（该事件将在下一节中讲解）。

【示例 10-6】 显式验证表单。

下面的示例使用 checkValidity()方法显式验证表单。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>使用 checkValidity()方法显式验证表单</title>
<script type="text/javascript">
function CheckForm(frm){
    if(frm.myEmail.checkValidity()){        /* 显式验证邮件 */
        alert("邮件格式正确！");
    }else{
        alert("邮件格式错误！");
    }
```

```
}
</script>
</head>
<body>
<div>
  <form action="" method="post">
    邮件:
    <input type="email" name="myEmail" id="myEmail" value="yxw740@163.com"
/>
    <br />
    <input type="submit" value="提交" onclick="return CheckForm(this.form)"
/>
  </form>
</div>
</body>
</html>
```

代码分析：示例 10-6 中，单击“提交”按钮时，会先调用 CheckForm()函数进行验证，再使用浏览器内置的验证功能进行验证。CheckForm()函数包含了 checkValidity()方法的显式验证。在使用 checkValidity()进行显式的验证时，还会触发所有的结果事件和 UI 触发器，就好像表单提交了一样。

2．setCustomValidity()方法

自定义错误提示信息的方法。

当默认的提示错误满足不了需求时，可以通过该方法自定义错误提示。当通过此方法自定义错误提示信息时，元素的 validationMessage 属性值会更改为定义的错误提示信息，同时 ValidityState 对象的 customError 属性值变成 true。下面通过示例了解其使用方法。

【示例 10-7】 自定义错误提示信息。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>自定义错误提示信息</title>
<script type="text/javascript">
function CheckForm(frm){
    var name=frm.name;
    if(name.value==""){
        name.setCustomValidity("请填写您的姓名！");   /* 自定义错误提示 */
    }else{
        name.setCustomValidity("");                  /* 取消自定义错误提示 */
    }
}
</script>
</head>
<body>
<div>
  <form action="" method="post">
    姓名:
    <input id="name" name="name" placeholder="First and Last Name" required
/>
    <input type="submit" value="提交" onClick="CheckForm(this.form)" />
  </form>
</div>
</body>
</html>
```

代码分析：在示例 10-7 中，在提交表单时，如果姓名为空，则自定义一个提示信息；如果姓名不为空，则取消自定义错误信息。

10.4.5　表单验证的事件

HTML 5 为我们提供了一个表单验证的事件。下面介绍 invalid 事件。

表单元素为通过验证时触发。无论是提交表单还是直接调用 checkValidity 方法，只要有表单元素没有通过验证，就会触发 invalid 事件。invalid 事件本身不处理任何事情，我们可以监听该事件，自定义事件处理。

【示例 10-8】 监听 invalid 事件。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>监听 invalid 事件</title>
<script type="text/javascript">
function invalidHandler(evt){
    //获取当前被验证的对象
    var validity = evt.srcElement.validity;
    //检测 ValidityState 对象的 valueMissing 属性
    if(validity.valueMissing){
        alert("姓名是必填项，不能为空")
    }
    //如果不希望看到浏览器默认的错误提示方式，可以使用下面的方式取消
    evt.preventDefault();
}
window.onload=function(){
    var name=document.getElementById("name");
    //注册监听 invalid 事件
    name.addEventListener("invalid",invalidHandler,false);
}
</script>
</head>
<body>
<div>
  <form action="" method="post">
    姓名：
    <input id="name" name="name" placeholder="First and Last Name" required
/>
    <input type="submit" value="提交" />
  </form>
</div>
</body>
</html>
```

代码分析：在示例 10-8 中，页面初始化时，为姓名输入框添加了一个监听的 invalid 事件。当表单验证没通过时，会触发 invalid 事件，invalid 事件会调用注册到事件里的函数 invalidHandler()。这样就可以在自定义的函数 invalidHandler()中做任何事情了。

一般情况下，在 invalid 事件处理完成后，还是会触发浏览器默认的错误提示。必要的时候，可以屏蔽浏览器后续的错误提示，可以使用事件的 preventDefault()方法，阻止浏览器的默认行为，并自行处理错误提示信息。

通过使用 invalid 事件使得表单开发更加灵活。如果需要取消验证，可以使用前面讲过的 novalidate 特性。

10.5　实验室：用户注册页面

本节将使用本章所讲述的 HTML 5 表单技术，设计一个简单的用户注册页面。由于 HTML 5 的表单不但增加了很多特性，而且表单的验证方式也有很大的转变，因此本节介绍的案例的实现，也将突破传统的思维方式。

1. 案例简介

本节介绍的案例是一个用户注册页面，页面内容包括：姓名、年龄、邮箱、微博地址、毕业时间、考试成绩、所在国家等。使用 HTML 5 表单技术实现该注册页面与传统的实现方式相比有非常大的差异，主要表现在以下几个方面。

- 部分表单元素提供了描述性的信息，输入框获取焦点时，提示信息会消失。
- 使用了 HTML 5 新增的表单类型，其中年龄表现为滑块，邮件和微博会自带验证格式，毕业时间与考试成绩的表单元素是可以微调的。
- 国家的选择会有提供的下拉提示，是使用 list 特性和 datalist 元素完成的。
- 年龄的滑块型的表单元素不显示数值，使用 output 元素协助其显示。
- 表单的验证完全突破了传统的验证方式。

用户注册页面的效果如图 10-5 所示，下面逐步实现它。



图 10-5　HTML 5 实现的用户注册页面

2. 页面基本元素

页面内容包括：姓名、年龄、邮箱、微博、毕业时间、考试成绩、国家等，其中年龄、

邮箱、微博、毕业时间、考试成绩使用了新型的表单类型，各种新型的表单都包含自身的验证信息。页面元素信息如示例 10-9 所示。另外，这里为每一个表单元素增加了 title 特性，其值为表单的文字名称，将用于后续的表单验证提示。

【示例 10-9】 用户注册页面。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<style type="text/css">
  … / * 样式表省略 */
</style>
</head>
<body>
<form id="register" name="register">
  <label for="firstName"><span>姓名</span>
    <input name="firstName" type="text" title="姓名" placeholder="请输入您
    的姓名" required />
  </label>
  <label for="age"><span>年龄</span>
<input id="age" name="age" type="range" min="0" max="99" step="1" value="20"
onchange="displayAage.value=this.value" />
    <output name="displayAage">20</output>
  </label>
  <label for="emailaddress"><span>邮箱</span>
    <input type="email" name="emailaddress" title="邮箱" placeholder="请输
    入您的邮箱" required />
  </label>
  <label for="weibo"><span>微博</span>
    <input type="url" name="weibo" title="微博" placeholder="请输入您的微博
    地址" required />
  </label>
  <label for="graduation"><span>毕业时间</span>
    <input name="graduation" type="date" title="毕业时间" placeholder="请输
    入您的毕业时间" />
  </label>
  <label for="score"><span>考试成绩</span>
    <input name="score" type="number" title="考试成绩" placeholder="请输入您
    的考试成绩" min="0" max="10" step="0.2" />
  </label>
  <label for="country"><span>国家</span>
    <input name="country" type="text" title="国家" list="countries" />
  </label>
  <datalist id="countries">
    <option value="Australia">Australia</option>
    <option value="Germany">Germany</option>
    <option value="United Kingdom">United Kingdom</option>
    <option value="United States">United States </option>
  </datalist>
  <input type="submit" name="submit" value="提交" onclick="clearError()" />
  <div id="error"></div>
</form>
</body>
</html>
```

3．实现表单验证

表单验证将突破传统方式。如示例 10-9 所示，表单提交时，没有专门的表单验证处理函数。HTML 5 的表单验证会在提交表单之前做好准备工作，如示例中的可用与验证的表单类型和表单特性等。下面是其他准备工作。

在页面加载完成后，将执行下面的 JavaScript 脚本，主要做两件事情：一件是注册表单的 invalid 事件，当表单验证时，如果捕获到错误会立即触发；另一件是为年龄项自定义提示信息。

```
<script type="text/javascript">
window.onload=function(){
    var register=document.getElementById("register");
    //注册监听表单中的 invalid 事件，捕获到错误即处理
    register.addEventListener("invalid",invalidHandler,true);
    //为年龄项添加自定义错误提示信息
    document.getElementById("age").setCustomValidity("年龄不能通过验证！");
}
</script>
```

4．其他处理函数

函数 addError()和 clearError()是处理提示信息的，分别表示添加提示和清除提示。

函数 invalidHandler()是用于处理 invalid 事件的。当表单验证有错误时，会立即触发 invalid 事件，并执行函数 invalidHandler()，该函数为了获得更加准确的错误信息，对 ValidityState 对象的各个属性分别进行了判断，并且屏蔽了系统的验证提示。

```
<script type="text/javascript">
function addError(err){
      document.getElementById("error").innerHTML += "* "+err+"<br />";
}
function clearError(){
      document.getElementById("error").innerHTML = "";
}
function invalidHandler(evt){
      //获取出错元素的 ValidityState 对象
      var validity = evt.srcElement.validity;
      var str="";
      //如果有自定义的提示信息，则使用它来提示
      if(validity.customError){
          str = evt.srcElement.validationMessage;
      }else{
          //以下是检测的 ValidityState 对象的各个属性，以判断具体错误
          if(validity.valueMissing){
                str+="不能为空；";
          }
          if(validity.typeMismatch){
                str+="与类型不匹配；";
          }
          if(validity.patternMismatch){
                str+="与 pattern 正则不匹配；";
          }
          if(validity.tooLong){
                str+="字符过长；";
```

```
            }
            if(validity.rangeUnderflow){
                  str+="不能小于最小值；";
            }
            if(validity.rangeOverflow){
                  str+="不能大于最大值；";
            }
            if(validity.stepMismatch){
                  str+="不符合 step 特性所推算出的规则；";
          }
          //使用表单元素的 title 特性值来组合提示信息
          str = evt.srcElement.title + str;
     }
     //添加错误提示
     addError(str);
     //阻止事件冒泡
     evt.stopPropagation();
     //取消后续的浏览器默认的处理方式
     evt.preventDefault();
}
</script>
```

至此，用户注册页面已经完成。此时，若单击“提交”按钮，表单仍然会自动验证。如果出现错误，会优先使用 invalid 事件的 invalidHandler()函数来做后续的错误处理。

10.6　小　　结

本章主要讲解了 HTML 5 的表单技术。重点讲解了新增的表单类型、新增的表单特性及元素和表单验证 API，其中分别对表单验证 API 的属性、方法和事件进行了重点的讲解，对涉及的 ValidityState 对象也做了逐一的剖析。本章的难点在于表单验证方式的思维转变，需要对表单验证的整个过程有非常充分的理解。下一章将介绍 HTML 5 的拖放功能。

10.7　习　　题

【习题 1】HTML 5 大幅度地改进了 input 元素的类型，以下哪些类型是新增的（多选）：

A．url 类型　　B．email 类型　　C．range 类型

D．number 类型　　E．tel 类型　　F．file 类型

【习题 2】HTML 5 增加了很多新的表单特性及元素，请选择哪些是新增的元素（多选）：

A．required　　B．placeholder　　C．autocomplete

D．datalist　　E．keygen　　F．output

【习题 3】ValidityState 对象有 8 个属性，分别针对 8 个方面的错误验证。请列出这 8 个属性并附加简要的说明。

【习题 4】请设计一个用户注册的表单，并使用 ValidityState 对象来验证表单。

第 11 章　可触到的拖放功能

在 Web 应用中，良好的用户体验是设计师们一直的追求，其中的拖放体验就是其中之一。在 HTML 5 之前，已经可以使用事件 mousedown、mousemove 和 mouseup 巧妙地实现页面内的拖放操作，但是拖放的操作范围只是局限在浏览器内部。HTML 5 提供的拖放 API，不但能直接实现拖放操作，而且拖放的范围已经超出浏览器的边界；HTML 5 提供的文件 API，支持拖放多个文件并上传。这一革命性的支持，为移动互联网应用进一步铺平了道路。本章就深入探讨拖放 API 和文件 API。

11.1　拖放 API

HTML 5 的拖放 API 基本包括三个方面：首先是为页面元素提供了拖放特性；其次是为鼠标事件增加了拖放事件；最重要的是提供了用于存储拖放数据的 DataTransfer 对象。下面分三个方面进行讲解。

11.1.1　新增的 draggable 特性

draggable 特性用于定义元素是否允许用户拖放，该特性有三个可选值：true、false 和 auto。通常大部分的页面元素是不可拖放的，如果要把元素变成可以拖放的，则可以设置 draggable 特性如下：

```
<div draggable="true"> </div>
```

另外，img 元素和 a 元素（需指定 href）默认是可以拖放的。

11.1.2　新增的鼠标拖放事件

为了使拖放控制更加具体，HTML 5 提供了 7 个与拖放相关的鼠标响应事件，而这 7 个事件会响应在不同的元素上。下面我们按照响应的时间先后顺序逐个介绍。

1. dragstart事件

开始拖放时触发的事件，事件的作用对象是被拖放的元素。

2. drag事件

拖放过程中触发的事件，事件的作用对象是被拖放的元素。

3．dragenter事件

有拖放的元素进入本元素的范围内时触发，事件的作用对象是拖放过程中鼠标经过的元素。

4．dragover事件

有拖放的元素正在本元素的范围内移动时触发，事件的作用对象是拖放过程中鼠标经过的元素。

5．dragleave事件

有拖放的元素离开本元素的范围时触发，事件的作用对象是拖放过程中鼠标经过的元素。

6．drop事件

有拖放的元素被拖放到本元素中时触发，事件的作用对象是拖放的目标元素。

7．dragend事件

拖放操作结束时触发，事件的作用对象是被拖放的元素。

在拖放过程中，我们就可以触发这 7 个拖放事件，来实现页面的灵活控制。

11.1.3　DataTransfer 对象

HTML 5 提供了 DataTransfer 对象，用以支持拖放数据的存储。使用拖放的目的，就是希望在拖放的过程中，有数据交换，而 DataTransfer 对象就充当了这种媒介。DataTransfer 对象有其自身的属性和方法，可以完成对拖放的数据的各种处理。

1．dropEffect属性

设置或获取拖放操作的类型和要显示的光标类型。如果该操作效果与起初设置的 effectAllowed 效果不符，则拖放操作失败。可以修改设置，包含这几个值：none、copy、link 和 move。

2．effectAllowed属性

设置或获取数据传送操作可应用于该对象的源元素。可以指定值为 none、copy、copyLink、copyMove、link、linkMove、move、all 和 uninitialized。

3．types属性

获取在 dragstart 事件触发时为元素存储数据的格式，如果是外部文件的拖放，则返回 Files。

4．files属性

获取存储在 DataTransfer 对象中的正在拖放的文件列表 FileList，可以使用数组的方式

去遍历。

5．clearData()方法

清除 DataTransfer 对象中存放的数据。语法如下：

```
clearData([sDataFormat])
```

参数说明：

[sDataFormat]为可选参数。参数可取值为：Text、URL、File、HTML、Image，即可删除指定格式的数据。如果该参数省略，则清除全部数据。

6．setData()方法

向内存中的 DataTransfer 对象添加指定格式的数据。语法如下：

```
setData([sDataFormat], [data])
```

参数说明：[sDataFormat]为数据参数类型，可取值为：Text、URL。[data]数据为字符串或 url 地址。

7．getData()方法

从内存中的 DataTransfer 对象中获取数据。语法如下：

```
getData([sDataFormat])
```

参数说明：[sDataFormat]为数据参数类型，可取值为：Text、URL。

8．setDragImage()方法

设置拖放时，跟随鼠标移动的图片。语法如下：

```
setDragImage([imgElement], [x], [y])
```

参数说明：[imgElement]：表示图片对象；[x]、[y]：分别表示相对于鼠标位置的横坐标和纵坐标。

9．addElement()方法

添加一起跟随拖放的元素，如果你想让某个元素跟随被拖放元素一同被拖放，则使用此方法。语法如下：

```
addElement([element])
```

参数[element]表示一起跟随拖放的元素对象。

11.1.4　实验室：拖放元素的内容

下面使用前面学习的拖放 API，来实现拖放页面中元素的内容，即把一个页面元素里的内容通过拖放的方式，在另一个元素内产生一个复制，实现效果如图 11-1 所示。



图 11-1　拖放页面元素的内容

下面分 4 个步骤来介绍拖放的实现。

1. 设计页面元素

在页面中添加两个元素，分别作为拖放的源元素和目标元素，并设置样式表。其中，源元素内部包含文字和图片内容，目标元素的内容为空；源元素设置 draggable 特性值为 true（draggable="true"），表示源元素是可以被拖放的。页面设置如下所示。

【示例 11-1】 拖放页面元素的内容。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>拖放页面的内容</title>
<style type="text/css">
… / * 样式表省略 */
</style>
</head>
<body>
<!-- 源元素 dragSource -->
<div id="dragSource" draggable="true">拖这里<img src="images/icon6.png"
width="75" height="72"></div>
<!-- 目标元素 dropTarget -->
<div id="dropTarget"></div>
</body>
</html>
```

2. 添加ondragstart监听事件

给拖放的源元素添加 ondragstart 监听事件，事件触发时，把源元素里的内容追加至 dataTransfer 对象中。最后把添加监听事件的处理函数 DragStart()追加至 window.onload 事件中。

```
<script type="text/javascript">
function DragStart(){
     var source = document.getElementById('dragSource');   /* 拖放源元素 */

     /* 监听 dragstart 事件：作用在源元素上 */
     source.addEventListener('dragstart', function(e) {
         e.dataTransfer.setData('text/plain', e.target.innerHTML);
                                         /* 向 dataTransfer 对象中追加数据 */
         e.dataTransfer.effectAllowed="copy";
     },false);
}
/* 添加函数 DragStart 值 window.onload 监听事件 */
window.addEventListener('load', DragStart,false);
</script>
```

3. 添加dragover监听事件

给拖放的目标元素添加 dragover 监听事件，事件触发时，改变目标元素的样式，并屏蔽了浏览器的默认处理事件。最后把添加监听事件的处理函数 DragOver()，追加至 window.onload 事件中。

目标元素的 preventDefault()方法可以取消浏览器的默认处理，否则，无法实现拖放。

```
<script type="text/javascript">
function DragOver(){
    var target = document.getElementById('dropTarget');   /* 拖放目标元素 */
    /* 监听 dragover 事件：作用在目标元素上 */
    target.addEventListener('dragover', function(e) {
        this.className="dragover";               /* 鼠标拖放经过时的样式 */
        e.preventDefault();                      /* 取消浏览器默认处理 */
    },false);
}
/* 添加函数 DragStart 值 window.onload 监听事件 */
window.addEventListener('load', DragOver,false);
</script>
```

4. 添加ondrop监听事件

给拖放的目标元素添加 ondrop 监听事件，事件触发时，获取 dataTransfer 对象中的数据，并追加到目标元素中，同时还原了样式。最后把添加监听事件的处理函数 Drop ()，追加至 window.onload 事件中。

```
<script type="text/javascript">
function Drop(){
    var target = document.getElementById('dropTarget');   /* 拖放目标元素 */
    /* 监听 drop 事件：作用在目标元素上 */
    target.addEventListener('drop', function(e) {
        var data=e.dataTransfer.getData('text/plain');
                                       /* 取得 dataTransfer 对象中的数据 */
        this.innerHTML += data;
        e.dataTransfer.dropEffect="copy";
        this.className="";             /* 还原样式 */
    },false);
}
```

```
/* 添加函数 DragStart 值 window.onload 监听事件 */
window.addEventListener('load', Drop,false);
</script>
```

至此，拖放页面元素的内容已经实现了。此时，可以把源元素中的内容拖放至目标元素中。

11.2　文件 API

HTML 5 提供了一个关于文件操作的文件 API，我们可以通过编程的方式选择和访问文件数据，使得从 Web 网页上访问本地文件系统变得十分简单。文件 API 主要涉及 FileList 对象、File 对象、Blob 接口和 FileReader 接口。

11.2.1　新增的标签特性

HTML 5 仍然沿用传统的文件上传方式，借助 file 类型的表单元素来实现文件的上传。与之前不同的是，HTML 5 为 file 类型的表单元素新增了 multiple 特性和 accept 特性。

1．multiple特性

multiple 特性可允许同时选择多个上传文件。在 HTML 5 之前，file 类型的表单元素只允许选择一个上传文件。而在 HTML 5 中，可借助 multiple 特性同时选择多个上传文件。multiple 特性的使用方法如下：

```
<input type="file" multiple />
```

同时选择多个上传文件，得到的是一个 FileList 对象，该对象是一个 File 对象的列表。关于这两个对象会在下一节讲述。

2．accept特性

accept 特性规定了可通过文件上传提交的文件类型。HTML 5 规范试图使用 accept 特性限制文件上传只能接受指定的文件类型，但目前各个主流浏览器并没有做这样的限制，仅实现了在打开文件窗口时，默认选择指定的文件类型。accept 特性使用方法如下：

```
<input type="file" accept="image/gif" />
```

这行代码说明了只接受 gif 格式的图片，实际中在选择上传文件时，默认仅显示 gif 格式的文件。当然也可以选择其他类型的文件进行上传，没有实际的限制。

11.2.2　FileList 对象与 File 对象

当用户在 file 类型的表单元素中同时选择多个文件时，可通过编程的方式获得一个文件列表，即 FileList 对象。FileList 对象里的每一个文件又是一个 File 对象。

FileList 对象是 File 对象的一个集合，可使用数组的方式遍历 FileList 对象里的每一个

File 对象。下面通过一个示例进一步了解这两个对象之间的关系。

【示例 11-2】 遍历 FileList 对象。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>遍历 FileList 对象</title>
<script type="text/javascript">
function ShowFiles(){
      var fileList=document.getElementById("files").files;
                                          /* 获取 FileList 对象 */
     var msg=document.getElementById("msg");
     var file;
     for(var i=0;i<fileList.length;i++){
           file=fileList[i];              /* 获取单个 File 对象 */
           msg.innerHTML+=file.name+";<br />";
    }
}
</script>
</head>
<body>
<form action="" method="post">
  <input type="file" id="files" multiple />  <!-- 可选择多个文件 -->
  <input type="button" value="显示文件" onclick="ShowFiles()" />
  <p id="msg"></p>
</form>
</body>
</html>
```

代码分析：在示例 11-2 中，设置 file 类型的表单元素的特性 multiple，可选择多文件，可获取 FileList 对象。单击“显示文件”按钮，会执行函数 ShowFiles()，并把 FileList 对象内的所有 File 对象名称显示出来。

11.2.3　Blob 对象

Blob 接口代表原始二进制数据，通过 Blob 对象的 slice()方法，可以访问里面的字节数据。Blob 接口还有两个属性：size 和 type。

1. size属性

表示 Blob 对象的字节长度。Blob 对象的二进制数据可借助 FileReader 接口读取。如果 Blob 对象没有字节数，则 size 属性为 0。

2. type属性

表示 Blob 对象的 MIME 类型，如果是未知类型，则返回一个空字符串。使用 type 属性获取文件的 MIME 类型，可以更加精确地确定文件的类型，可避免因更改文件的扩展名而造成文件类型的误判。

3．slice()方法

使用 slice()方法可以实现文件的切割，并返回一个新的 Blob 对象。

4．File对象与Blob对象

File 对象继承了 Blob 对象，所以 File 对象也可以使用 Blob 对象的属性和方法。

5．示例介绍

File 对象可以像 Blob 对象一样去使用 size 属性和 type 属性。下面的示例就使用这两个属性获取上传文件的基本数据信息。

【示例 11-3】 获取文件的大小和类型。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>获取文件的大小和类型</title>
<script type="text/javascript">
function ShowType(){
       var files=document.getElementById("files").files;/* 获取 FileList 对
       象 */
       var msg=document.getElementById("msg");
       var file;
       for(var i=0;i<files.length;i++){
            file=fileList[i];                /* 获取单个 File 对象 */
            msg.innerHTML+="字节长度："+file.size+";<br />";
            msg.innerHTML+="文件类型："+file.type+";<br />";
    }
}</script>
</head>
<body>
<form action="" method="post">
  <input type="file" id="files" multiple accept="image/*" />
  <input type="button" value="显示文件数据" onclick="ShowType()" />
  <p id="msg"></p>
</form>
</body>
</html>
```

代码分析：在示例 11-3 中，遍历每一个上传的 File 对象，并输出每个 File 对象的文件大小（size 属性）和文件类型（type 属性）。当然，还可以根据获取的文件大小和文件类型，做一些判断或限制。

11.2.4　FileReader 接口

FileReader 接口提供了一些读取文件的方法与一个包含读取结果的事件模型。作为 File API 的一部分，FileReader 接口主要是把文件读入内存，并读取文件中的数据。

由于部分早期版本的浏览器没有实现 FileReader 接口，因此在使用 FileReader 接口之前，有必要检测一下浏览器支持的情况。检测方法如下：

```
if(typeof FileReader == "undefined"){
     alert("浏览器未实现 FileReader 接口");
}else{
     var reader = new FileReader();
}
```

1. 接口的属性

FileReader 接口拥有三个属性，分别用于返回读取文件的状态、数据和读取时发生的错误。

- readyState 属性（只读）

获取读取文件的状态。该状态有如下 3 个值。

  - EMPTY（值为 0）：表示新的 FileReader 接口已经构建，且没有调用任何读取方法时的默认状态。
  - LOADING（值为 1）：表示有读取文件的方法正在读取 File 对象或 Blob 对象，且没有错误发生。
  - DONE（值为 2）：表示读取文件结束。可能整个 File 对象或 Blob 对象已经完全读入内存中，或者在文件读取的过程中出现错误，或者读取过程中使用了 abort()方法强行中断。

- result 属性（只读）

获取已经读取的文件数据。如果是图片，将返回 base64 格式的图片数据。

- error 属性（只读）

获取读取文件过程中出现的错误。该错误包含如下 4 种类型。

  - NotFoundError：找不到读取的资源文件。FileReader 接口会返回 NotFoundError 错误，同时读取文件的方法也会抛出 NotFoundError 错误异常。
  - SecurityError：发生安全错误。FileReader 接口会返回 SecurityError 错误，同时读取文件的方法也会抛出 SecurityError 错误异常。
  - NotReadableError：无法读取的错误。FileReader 接口会返回 NotReadableError 错误，同时读取文件的方法也会抛出 NotReadableError 错误异常。
  - EncodingError：编码限制的错误。通常是数据的 URL 表示的网址长度受到限制。

2. 接口的方法

FileReader 接口拥有 5 个方法，其中有 4 个用于读取文件，一个用来中断读取过程。

- readAsArrayBuffer()方法

将文件读取为数组缓冲区。该方法的语法如下：

```
readAsArrayBuffer( <blob> );
```

参数<blob>表示一个 Blob 对象的文件。readAsArrayBuffer()方法就是把该 Blob 对象的文件读取为数组缓冲区。

- readAsBinaryString ()方法

将文件读取为二进制字符串。该方法的语法如下：

```
readAsBinaryString( <blob> );
```

参数说明<blob>表示一个 Blob 对象的文件。readAsBinaryString()方法就是把该 Blob

对象的文件读取为二进制字符串。

- readAsText ()方法

将文件读取为二进制字符串。该方法的语法如下：

```
readAsText ( <blob> , <encoding> );
```

参数说明如下。

<blob>表示一个 Blob 对象的文件。readAsText()方法就是把该 Blob 对象的文件读取为文本。

<encoding>：表示文本的编码方式，默认值为 UTF-8。

- readAsDataURL ()方法

将文件读取为 DataURL 字符串。该方法的语法如下：

```
readAsDataURL ( <blob> );
```

参数<blob>表示一个 Blob 对象的文件。readAsDataURL()方法就是把该 Blob 对象的文件读取为 Data URL 字符串。

- abort()方法

用于中断读取操作。该方法的语法如下：

```
abort();
```

该方法没有参数。

3．接口的事件

- loadstart 事件

开始读取数据时触发的事件。

- progress 事件

正在读取数据时触发的事件。

- load 事件

成功完成数据读取时触发的事件。

- abort 事件

中断读取数据时触发的事件。

- error 事件

读取数据发生错误时触发的事件。

- loadend 事件

结束读取数据时触发的事件，数据读取可能成功也可能失败。

4．接口的方法示例

下面通过一个示例来演示 FileReader 接口的 4 个读取文件的方法。

【示例 11-4】 FileReader 接口的方法示例。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
```

```
<title>FileReader 接口的方法示例</title>
<script type="text/javascript">
//读取文件
function ReadAs(action){
     var blob=document.getElementById("files").files[0];
     if(blob){
          var reader = new FileReader();   /* 声明接口对象 */
          //根据参数 action，选择读取文件的方法
          switch (action.toLowerCase()){
               case "binarystring":
                    reader.readAsBinaryString(blob);
                                        /* 将文件读取为二进制字符串 */
                    break;
               case "arraybuffer":
                    reader.readAsArrayBuffer(blob);
                                        /* 将文件读取为数组缓冲区 */
                    break;
               case "text":
                    reader.readAsText(blob);      /* 将文件读取为文本 */
                    break;
                case "dataurl":
                    reader.readAsDataURL(blob);
                                        /* 将文件读取为 DataURL 数据 */
                    break;
        }
        reader.onload=function(e){
            //访问 FileReader 的接口属性 result，把读取到内存里的内容获取出来
            var result = this.result;
            //如果是图像文件，且读取内容为 DataURL 数据，那么就显示为图片
            if(/image\/\w+/.test(blob.type) && action.toLowerCase()=="
            dataurl"){
                document.getElementById("result").innerHTML = "<img src='"
                + result + "' />";
            }else{
                document.getElementById("result").innerHTML = result;
            }
        }
    }
}
</script>
</head>
<body>
<form action="" method="post">
  <input type="file" id="files" multiple accept="image/*" />
  <input type="button" value="读取为数组缓存区" onclick="ReadAs('Array
  Buffer')" />
  <input type="button" value="读取为二进制" onclick="ReadAs('BinaryString')"
/>
  <input type="button" value="读取为文本" onclick="ReadAs('Text')" />
  <input type="button" value="读取为图像" onclick="ReadAs('DataURL')" />
  <p id="result"></p>
</form>
</body>
</html>
```

运行结果如图 11-2 所示。

代码分析：在示例 11-4 中，使用 ReadAs()函数读取 file 类型的表单元素选择的文件。

在 ReadAs()函数中，通过参数 action 的值，选择不同的读取方法，分别实现了 FileReader 接口的 4 个读取文件的方法。在页面按钮中，我们通过传递不同的 action 参数，响应不同的读取文件的方法。如图 11-2 所示，我们选择了一个图片文件，然后单击各个按钮，页面会根据读取文件的方法的不同，显示不同形式的内容。

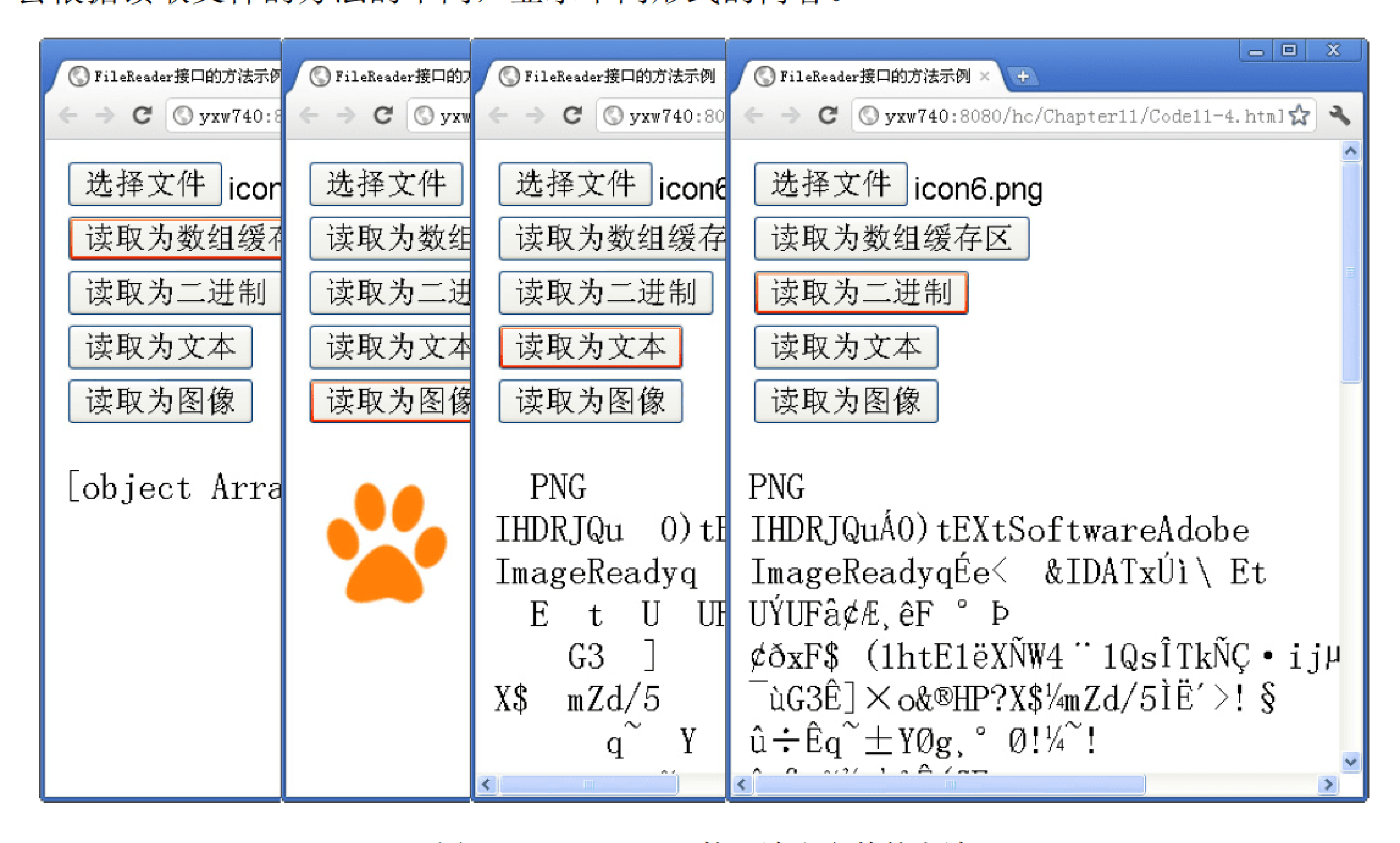


图 11-2　FileReader 接口读取文件的方法

5．接口的事件示例

FileReader 接口提供了 6 个事件，下面通过一个示例来查看各个事件的响应顺序。

【示例 11-5】 FileReader 接口的事件响应顺序。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>FileReader 接口的事件响应顺序</title>
<script type="text/javascript">
     var blob=document.getElementById("files").files[0];
     var message = document.getElementById("message");
     var reader = new FileReader();               /* 声明接口对象 */
     //添加 loadstart 事件
     reader.onloadstart=function(e){
         message.innerHTML+= "Event:loadstart;<br />";
     }
     //添加 progress 事件
     reader.onprogress=function(e){
         message.innerHTML+= "Event:progress;<br />";
     }
     //添加 load 事件
     reader.onload=function(e){
         message.innerHTML+= "Event:load;<br />";
     }
```

```
    //添加 abort 事件
    reader.onabort=function(e){
        message.innerHTML+= "Event:abort;<br />";
    }
    //添加 error 事件
    reader.onerror=function(e){
        message.innerHTML+= "Event:error;<br />";
    }
    //添加 loadend 事件
    reader.onloadend=function(e){
        message.innerHTML+= "Event:loadend;<br />";
    }
    reader.readAsDataURL(blob);        /* 读取文件至内存 */
}</script>
</head>
<body>
<form action="" method="post">
  <input type="file" id="files" multiple accept="image/*" />
  <input type="button" value="读取文件" onclick="FileReaderEvent()" />
  <p id="message"></p>
</form>
</body>
</html>
```

运行结果如图 11-3 所示。

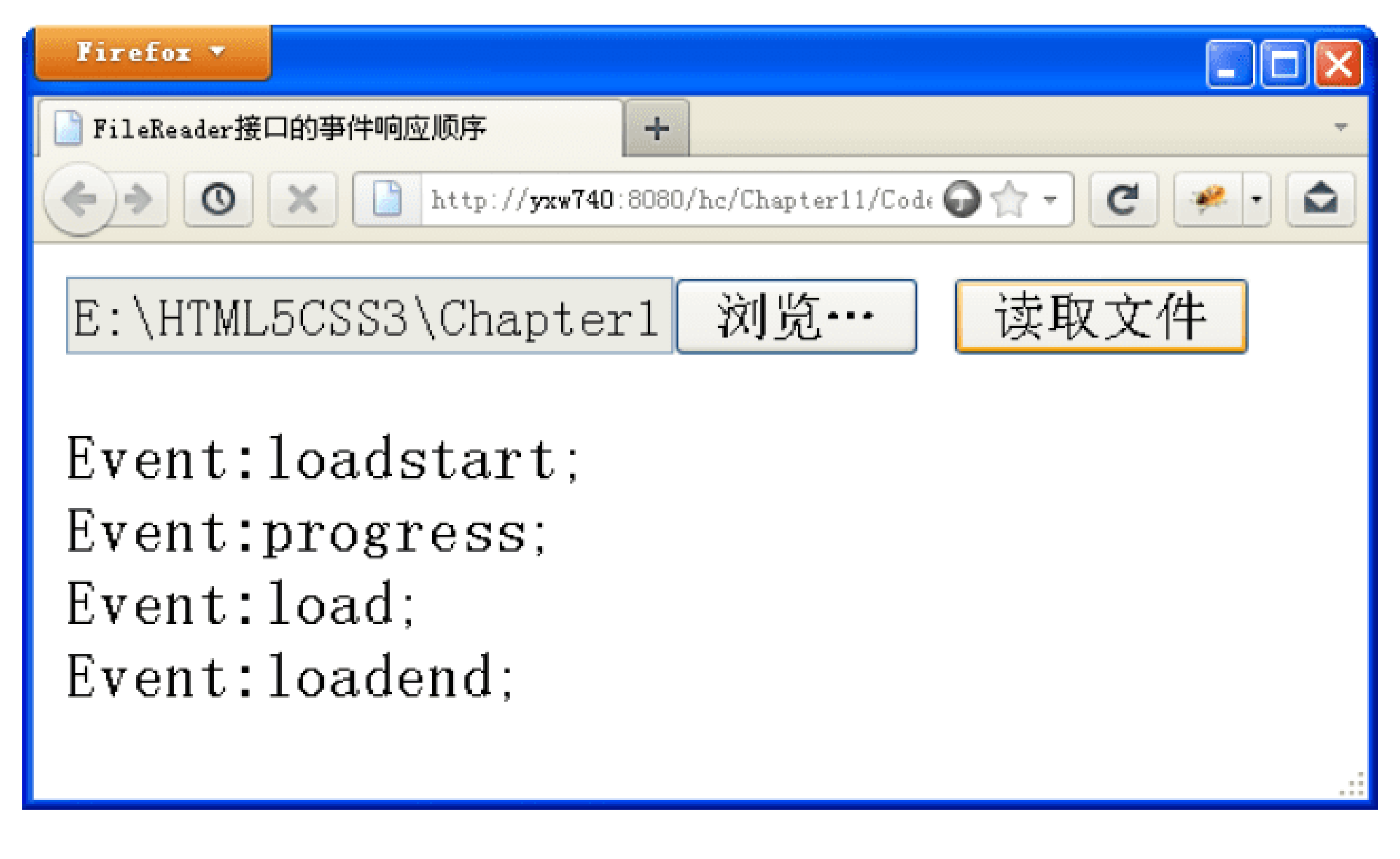


图 11-3　FileReader 接口在 Firefox 中的事件响应顺序

代码分析：在示例 11-5 中，为 FileReader 接口对象添加了所有的事件（6 个事件），每个事件的处理仅仅是输出事件的名称。如图 11-3 所示，当选择好文件并单击“读取文件”按钮时，页面会按照 FileReader 接口事件的响应顺序执行各个事件的输出。

11.3　实验室：把图片拖入浏览器

在 HTML 5 之前，很难实现超出浏览器边界的事情，如把电脑上的文件拖入浏览器，就很难实现。本节就用前面学习的拖放 API 和文件 API 来做这样的事情。

1. 案例简介

本节的案例是把电脑上的图片拖入浏览器的网页中的指定区域，并显示出来。实现效果如图 11-4 所示。

接下来，我们就一步一步地实现这个功能。

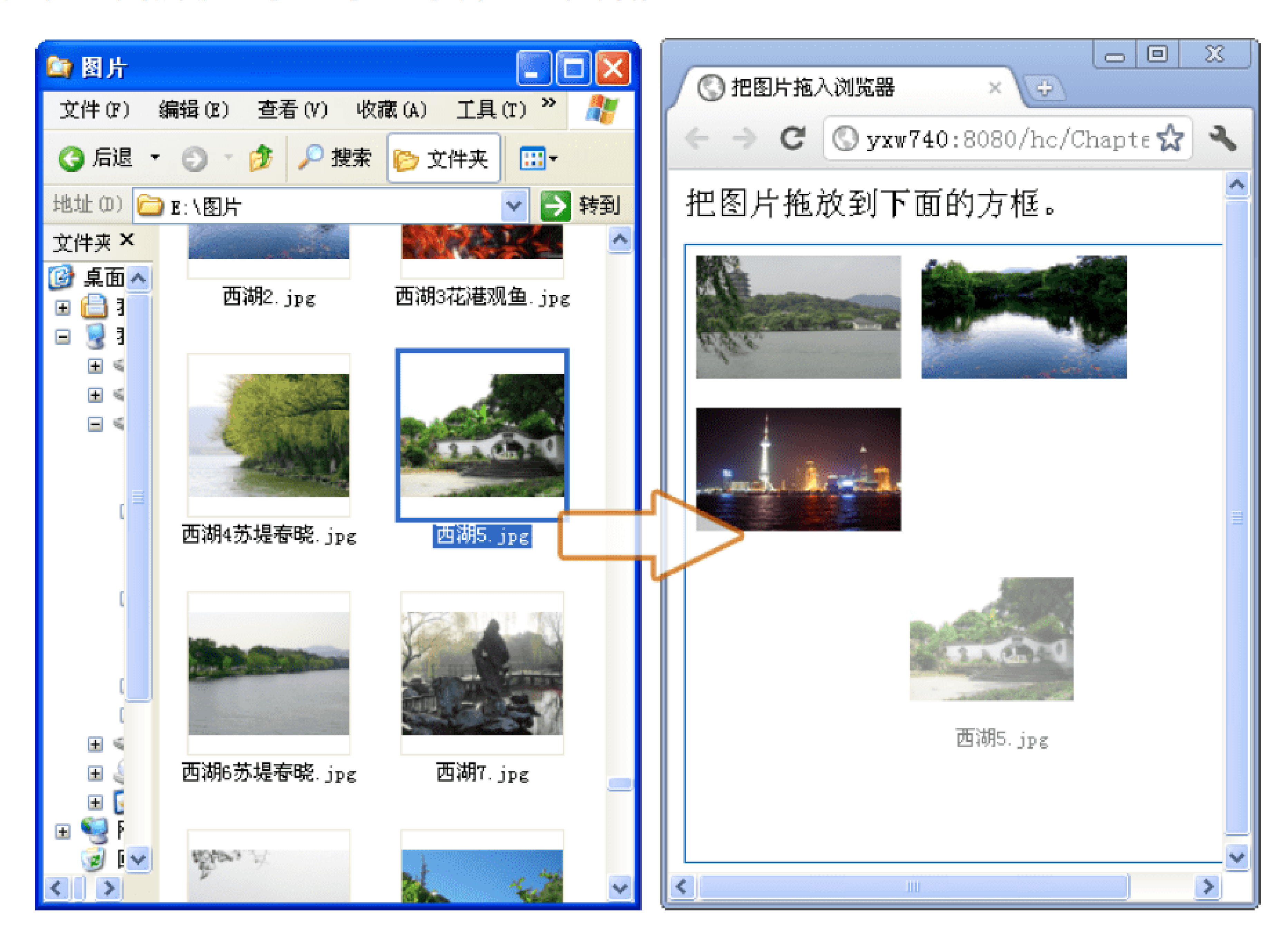


图 11-4　把电脑上的图片拖入浏览器

2. 设计网页基本元素

页面中，设计一个 div 元素，id 特性为 dropTarget，用于容纳拖进来的图片，是一个图片容器。页面中还针对此容器，添加了基本的样式表，如示例 11-6 所示。

【示例 11-6】 把图片拖入浏览器。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>把图片拖入浏览器</title>
<style type="text/css">
#dropTarget {
    width:300px;
    height:300px;
    margin:10px 0 0 0;
    border:1px solid #015EAC;
}
#dropTarget img {
    width:100px;
    height:60px;
    margin:5px;
}
</style>
</head>
```

```
<body>
<div>把图片拖放到下面的方框。</div>
<div id="dropTarget"></div>
</body>
</html>
```

3．基本函数的实现

首先定义一个全局的变量，表示图片容器的对象，方便各个函数访问。这里还定义了一个用于拖放的 drop 事件处理函数 dropHandle()和加载单个文件的函数 loadImg()，用于对拖放进来的图片文件进行处理。

```
<script type="text/javascript">
//定义目标元素的变量
var target;
//drop 事件处理函数
function dropHandle(e) {
      var fileList  = e.dataTransfer.files,    /*获取拖放的文件*/
      fileType;
      //遍历拖放的文件
      for(var i=0;i<fileList.length;i++){
           fileType = fileList[i].type;
           if (fileType.indexOf('image') == -1) {
                   alert('请拖放图片');
                   return;
           }
           //加载单个文件
           loadImg(fileList[i]);
   }
}
//加载指定的图片文件，并追加至 target 对象的元素中
function loadImg(file){
     //声明接口对象
     var reader = new FileReader();
     //添加 load 事件处理
     reader.onload = function(e)  {
         var oImg = document.createElement('img');
         oImg.src = this.result;    /* 获取读取的文件数据 */
         target.appendChild(oImg);
     }
     //读取文件
     reader.readAsDataURL(file);
}
</script>
```

4．页面加载处理

页面加载完成后，获取 target 目标容器，用于存放拖放进来的图片。给 target 容器添加 dragover 事件处理和 drop 事件处理，其中 drop 事件处理函数就是前面的脚本函数 dropHandle()。代码如下：

```
<script type="text/javascript">
window.onload = function() {
    //获取目标元素
    target = document.getElementById('dropTarget');
```

```
    //给目标元素添加 dragover 事件处理
    target.addEventListener('dragover', function(e) {
        e.preventDefault();
    }, false);
    //给目标元素添加 drop 事件处理，处理函数为 dropHandle()
    target.addEventListener('drop', dropHandle, false);
}
</script>
```

至此，文件拖放功能已经完全实现。如图 11-4 所示，可以直接把电脑里的图片文件拖入该页面的容器中。

11.4　小　　结

本章主要讲解了 HTML 5 的拖放 API 和文件 API。重点讲解了拖放 API 中用于拖放数据存储的 DataTransfer 对象和文件 API 中用于读取文件的 FileReader 接口，对于 File 对象和 Blob 对象，也要熟练掌握。本章的重点也是本章的难点，不论是 DataTransfer 对象还是 FileReader 接口，都提供了各自的属性、方法和事件，需要多加熟悉。Blob 对象不容易理解，也是本章的难点。

下一章将介绍 HTML 5 在本地存储方面的应用。

11.5　习　　题

【习题 1】如何把页面元素变成可以拖放的？

【习题 2】当有拖放的元素被拖放到本元素中时，会触发一个拖放事件，请问是哪个事件？

【习题 3】在 HTML 5 拖放的应用中，必然会发生数据交换，这种数据交换的媒介是什么？

【习题 4】在 HTML 5 中，允许同时选择多个上传文件（在文件域中增加 multiple 特性，即可选择多个文件），将会得到一个包含多个上传文件的对象，请问这个对象是哪一个（单选）：

A．DataTransfer 对象　　B．FileList 对象　　C．File 对象

D．Blob 对象　　E．FileReader 对象

【习题 5】FileReader 对象用于读取文件的数据，如何确定文件读取完毕？

第 12 章　本地存储让你的应用更加高效

随着 Web 应用的发展，需要在用户本地浏览器存储更多的应用数据，传统的使用 cookie 存储的方案已经不能满足发展的需求，而使用服务器端存储的方案则是一种无奈的选择。HTML 5 的 Web Storage API 是一个理想的解决方案，可以在客户端存储更多的数据，而且可以实现数据在多个页面中共享甚至是同步。如果是存储复杂的数据，则可以借助 Web SQL Database API 来实现，可以使用 SQL 语句完成复杂数据的存储与查询。本章将详细讲解这两个本地存储的方案。

12.1　本地存储对象——Web Storage

本节主要讲解 Web Storage API 的属性、方法和事件，以及如何利用此接口实现数据的存储和存储数据在不同页面间的通信等内容。

12.1.1　Web Storage 简介

Web Storage 的诞生与 cookie 功能的不足有着很大关系，下面对这方面的问题进行深入探讨。

1．cookie简介

Web 开发者都会记得 cookie 在客户端存储数据方面的优越表现，它甚至至今仍在作为主要的客户端存储来使用。

cookie 可用于在程序间传递少量的数据，对于 Web 应用来说，它是一个在服务器和客户端之间来回传送文本值的内置机制，服务器可以根据 cookie 来追踪用户在不同页面的访问信息。正因其卓越的表现，在目前的 Web 应用中，cookie 得到了最为广泛的应用。

尽管如此，cookie 仍然有很多不尽如人意的地方，主要表现在以下方面。

- 大小的限制：cookie 的大小被限制在 4KB。在 Web 的富应用环境中，不能接受文件或邮件那样的大数据。
- 带宽的限制：只要有涉及 cookie 的请求，cookie 数据都会在服务器和浏览器间来回传送。这样无论访问哪个页面，cookie 数据都会消耗网络的带宽。
- 安全风险：由于 cookie 会频繁地在网络中传送，而且数据在网络上是可见的，因此在不加密的情况下，是有安全风险的。
- 操作复杂：在客户端的浏览器中，使用 JavaScript 操作 cookie 数据是比较复杂的。但是服务器端可以很方便地操作 cookie 数据。

以上方面的不足，算是 cookie 的缺点，但也是优点，这要看在什么样的环境下，干什么用。一般在浏览器富应用的环境下，仅仅使用 cookie 是不足的。但对于较小的数据，且需要在服务器端和客户端频繁传送的时候，使用 cookie 的意义更大。

2. Web Storage简介

Web Storage 可以在客户端保存大量的数据，而且通过其提供的接口，访问数据也非常方便。然而，Web Storage 的诞生，并不是为了替代 cookie，相反，是为了弥补 cookie 在本地存储中表现的不足。

Web Storage 本地存储的优势主要表现在以下几个方面。

- 存储容量：提供更大的存储容量。在 Firefox、Chrome、Safari 和 Opera 中，每个网域为 5MB；在 IE8 及以上则每个网域为 10MB。
- 零带宽：Web Storage 中的数据仅仅是存储在本地，不会与服务器发生任何交互行为，所以不存在网络带宽的占用问题。
- 编程接口：Web Storage 提供了一套丰富的编程接口，使得数据操作更加方便。
- 独立的存储空间：每个域（包括子域）都有独立的存储空间，各个存储空间是完全独立的，因此不会造成数据的混乱。

由此可见，Web Storage 并不能完全替代 cookie，cookie 能做的事情，Web Storage 并不一定能做到，如服务器可以访问 cookie 数据，但是不能访问 Web Storage 数据。所以 Web Storage 和 cookie 是相互补充的，会在各自不同的方面发挥作用。

沿着移动互联网发展的路线，浏览器端的富应用是一种必然的趋势，而 Web Storage 作为完全的浏览器客户端的本地存储，将发挥越来越重要的作用。

12.1.2　localStorage 和 sessionStorage

1. sessionStorage和localStorage的区别

Web Storage 本地存储包括 sessionStorage（会话存储）和 localStorage（本地存储）。熟悉 Web 编程的人员第一次接触 Web Storage 时，会很自然地与 session 和 cookie 去对应。不同的是，cookie 和 session 完全是服务器端可以操作的数据，但是 sessionStorage 和 localStorage 则完全是浏览器客户端操作的数据。

sessionStorage 和 localStorage 完全继承同一个 Storage API，所以 sessionStorage 和 localStorage 的编程接口是一样的。下面的章节将会着重讲解编程接口 Storage API。

sessionStorage 和 localStorage 的主要区别在于数据存在的时间范围和页面范围，如表 12.1 所示。

表 12.1　sessionStorage和localStorage的区别

session Storage	localStorage
数据会保存到存储它的窗口或标签关闭时	数据的生命周期比窗口或浏览器的生命周期长
数据只在构建它们的窗口或标签页内可见	数据可被同源的每个窗口或者标签页共享

2．检测浏览器支持

在 HTML 5 的各项特性中，Web Storage 的浏览器支持度是比较好的。目前，所有的主流浏览器都在一定程度上支持 Web Storage。因而，Web Storage 成为 Web 应用中最安全的 API 之一。尽管如此，还是需要检查浏览器是否支持 Web Storage，因为在某种情况可能会导致浏览器不能使用 Web Storage 的功能。

【示例 12-1】 检测浏览器是否支持 Web Storage。

```
<script type="text/javascript">
function CheckStorageSupport(){
     //检测 sessionStorage
     if(window.sessionStorage){
         console.log("浏览器支持 sessionStorage 特性!");
     }else{
           console.log("浏览器不支持 sessionStorage 特性!");
     }
     //检测 localStorage
     if(window.localStorage){
           console.log("浏览器支持 localStorage 特性!");
    }else{
         console.log("浏览器不支持 localStorage 特性!");
    }
}
window.addEventListener("load",CheckStorageSupport,false);
</script>
```

在 Chrome 浏览器中的运行结果如图 12-1 所示。

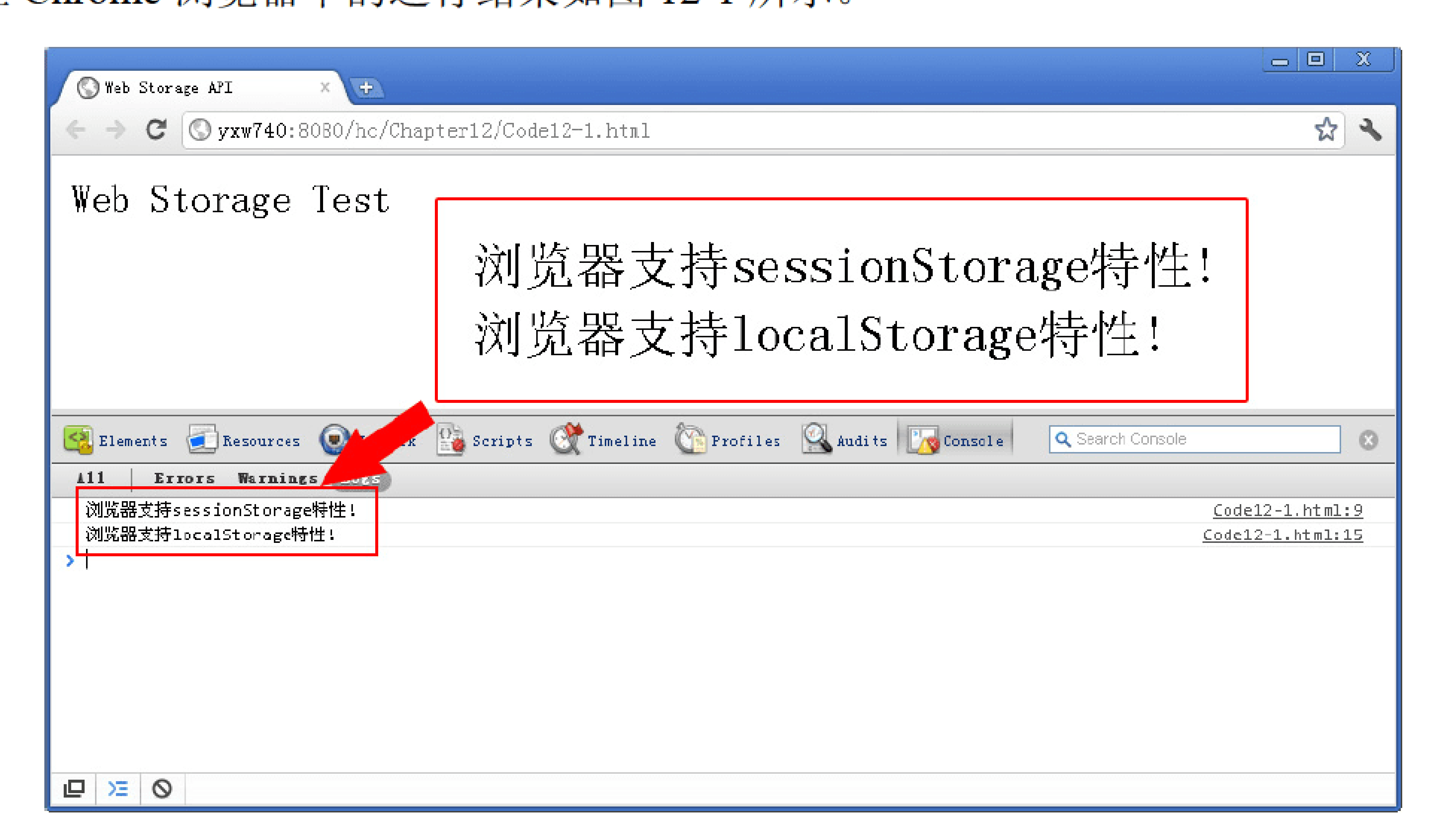


图 12-1　在控制台显示的检测结果

使用 console.log()方法，可以把调试的内容输出到浏览器的控制台。

12.1.3　设置和获取 Storage 数据

sessionStorage 和 localStorage 作为 window 的特性，完全继承 Storage API，它们提供

的操作数据的方法完全相同。下面以 sessionStorage 特性为例进行讲解。

1．保存数据到sessionStorage

sessionStorage 保存数据的完整语法如下：

```
window.sessionStorage.setItem("key","value");
```

参数说明“key”为字符串表示的“键”，“value”为字符串表示的“值”，setItem()表示保存数据的方法。

2．从sessionStorage中获取数据

如果知道保存到 sessionStorage 中的“键”，就可以取到对应的“值”。sessionStorage 获取数据的完整语法如下：

```
value = window.sessionStorage.getItem("key");
```

参数说明“key”和“value”分别表示“键”和“值”，与保存数据的键/值对应。getItem()为获取数据的方法。

3．设置和获取数据的其他写法

对于访问 Storage 对象还有更简单的方法，根据键/值的配对关系，直接在 sessionStorage 对象上设置和获取数据，可完全避免调用 setItem()和 getItem()方法。

保存数据的方法也可写成：

```
window.sessionStorage.key = "value";
```

或

```
window.sessionStorage["key"] = "value";
```

获取数据的方法，更加直接，可写成：

```
value = window.sessionStorage.key;
```

或

```
value = window.sessionStorage["key"];
```

这种灵活的使用方法，给编程带来极大的灵活性。当然，对于 localStorage 来说，同样具有上述设置数据和获取数据的方法。

4．示例介绍

【示例 12-2】 使用 sessionStorage 和 localStorage。

```
<script type="text/javascript">
function Test(){
     //在 localStorage 存储 localKey 的值为"localValue"
     window.localStorage.setItem("localKey","localValue");
     //获取存储在 localStorage 中的 localKey 的值，并输出到控制台
     console.log(window.localStorage.getItem("localKey"));
     //在 sessionStorage 存储 sessionKey 的值为"sessionValue"
```

```
    window.sessionStorage.setItem("sessionKey","sessionValue");
    ////获取存储在 sessionStorage 中的 sessionKey 的值，并输出到控制台
    console.log(window.sessionStorage.getItem("sessionKey"));
}
window.addEventListener("load",Test,false);
</script>
```

在 Chrome 浏览器中的运行结果如图 12-2 所示。

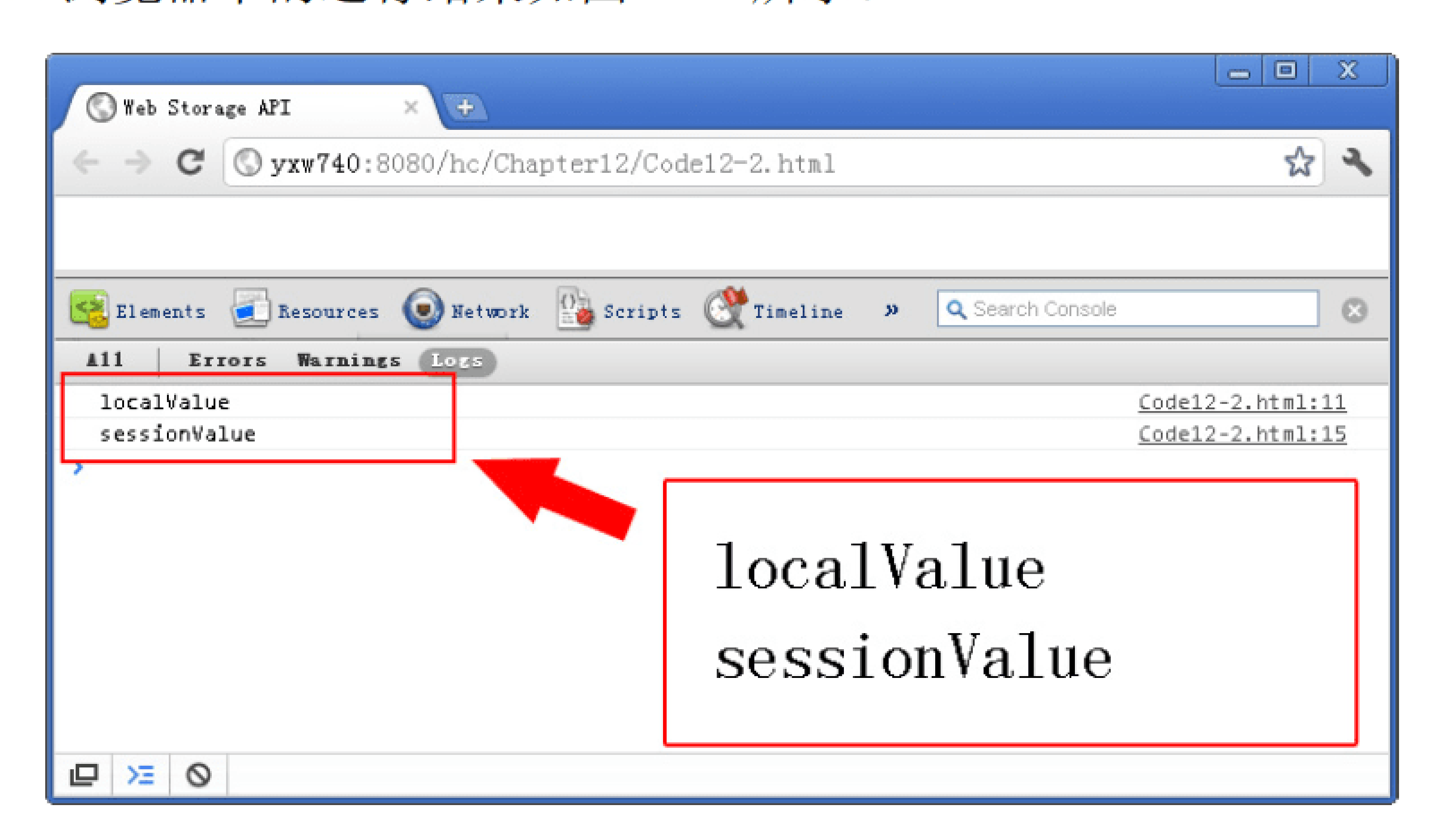


图 12-2　在控制台显示的输出结果

关于我们在 localStorage 和 sessionStorage 中存储的数据，可借助浏览器本身的功能进行查看，如在 Chrome 浏览器中，可在资源面板中查看存储的数据，如图 12-3 所示。

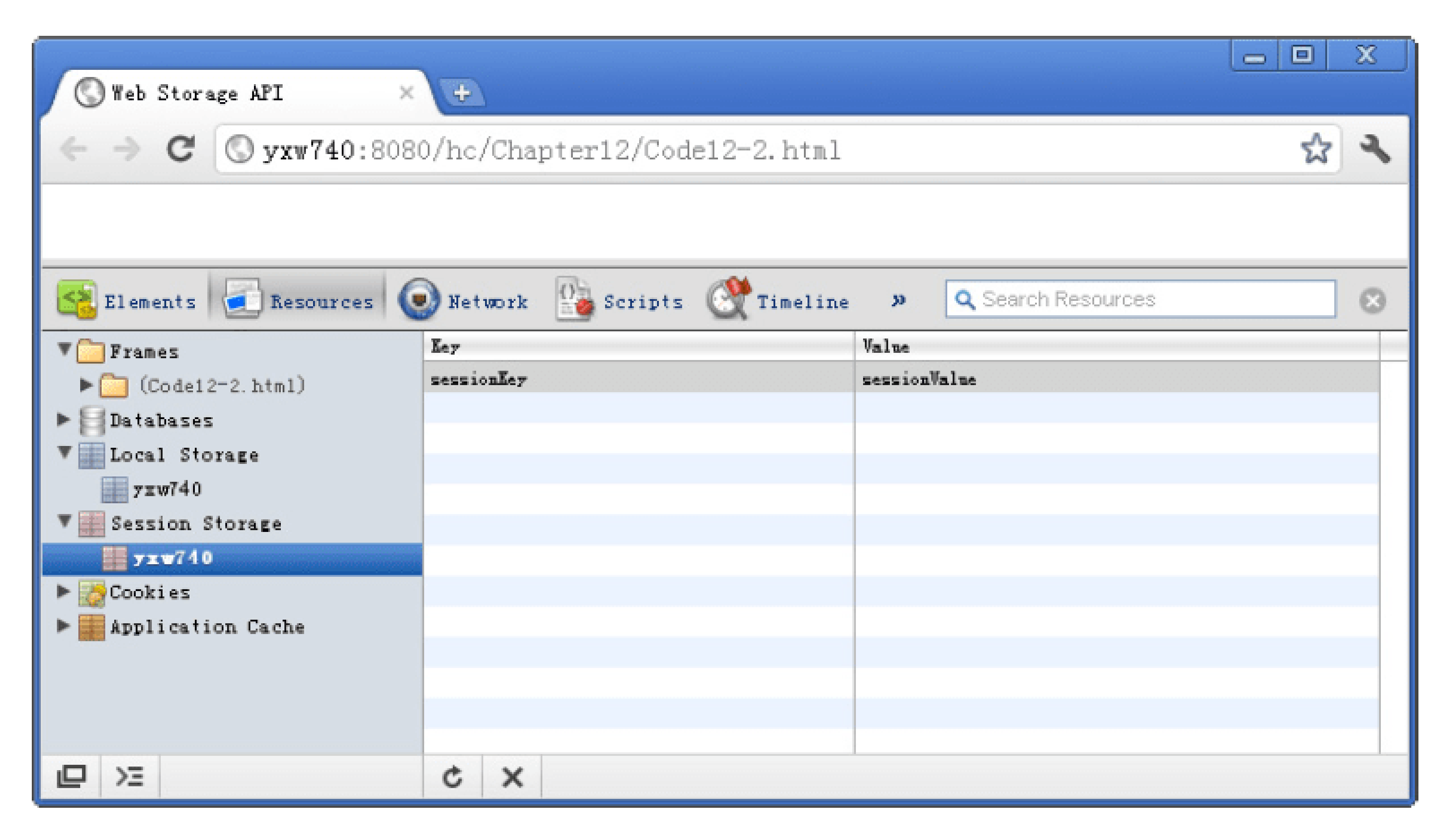


图 12-3　在 Chrome 浏览器中的资源面板中查看存储的数据

12.1.4　Storage API 的属性和方法

在上一节中学习了如何使用 setItem()方法存储数据，使用 getItem()方法获取数据。这些方法都来源于它们所继承的 Storage API 提供的方法。

1．Storage接口说明

【示例 12-3】 Storage 接口。

```
interface Storage {
  readonly attribute unsigned long length;
  DOMString? key(unsigned long index);
  getter DOMString getItem(DOMString key);
  setter creator void setItem(DOMString key, DOMString value);
  deleter void removeItem(DOMString key);
  void clear();
};
```

示例 12-3 中显示了接口中所有的属性和方法，下面进行详细介绍。

- length 属性：表示当前 Storage 对象中存储的键/值对的数量。Storage 对象是同源的，length 属性只能反映同源的键/值对数量。
- key(index)方法：获取指定位置的键。一般用于遍历某个 Storage 对象中所有的键，然后再通过键来取相应的值。参数 index 为从 0 开始的索引，最后一个索引是 length-1。
- getItem(key)方法：根据键返回相应的数据值。如果该键值存在，则返回值，否则返回 null。
- setItem(key,value)方法：将数据存入指定键对应的位置。如果对应的键值已经存在，则更新它。
- removeItem(key)方法：从存储对象中移除指定的键/值对。如果该键/值对存在，则移除它，否则不执行任何操作。
- clear()方法：清空 Storage 对象中所有的数据。如果 Storage 对象是空的，则不执行任何操作。

在使用 sessionStorage 和 localStorage 时，以上的属性和方法都可以使用，但需要注意其影响的范围。

2. 示例介绍

下面通过一个示例来了解 Storage 接口的应用。下面以 sessionStorage 为例进行介绍。

【示例 12-4】 使用 Storage 对象保存页面内容。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>使用 Storage 对象保存页面内容</title>
<script type="text/javascript">
//保存数据到 sessionStorage
function SaveStorage(frm){
     var storage = window.sessionStorage;
     storage.setItem("name",frm.name.value);
     storage.setItem("age",frm.age.value);
     storage.setItem("email",frm.email.value);
     storage.setItem("phone",frm.phone.value);
}
//遍历并显示 sessionStorage 中的数据
function Show(){
     var storage = window.sessionStorage;
     var result="";
     for(var i=0;i<storage.length;i++){
          var key = storage.key(i);                    /* 获取键 key */
```

```
            var value = storage.getItem(key);   /* 通过键 key 获取值 value */
            result += key + ":" + value + "; ";
      }
      /* 在指定的地方显示获取的存储内容 */
      document.getElementById("formdata").innerHTML = result;
}
</script>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
  <table width="100%" border="1" bordercolor="#CCCCCC" cellpadding="3"
cellspacing="0">
    <tr>
      <td>姓名</td>
      <td><input type="text" name="name" id="name" /></td>
    </tr>
    <tr>
      <td>年龄</td>
      <td><input type="text" name="age" id="age" /></td>
    </tr>
    <tr>
      <td>Email</td>
      <td><input type="text" name="email" id="email" /></td>
    </tr>
    <tr>
      <td>电话</td>
      <td><input type="text" name="phone" id="phone" /></td>
    </tr>
    <tr>
      <td></td>
      <td><input type="button" value="保存" onclick="SaveStorage(this.
      form)" />
        <input type="button" value="显示" onclick="Show()" /></td>
    </tr>
  </table>
</form>
<div id="formdata"></div>
</body>
</html>
```

运行结果如图 12-4 所示。

图 12-4　Storage 对象保存的页面内容

代码分析：在示例 12-4 中，有两个脚本处理函数 SaveStorage()和 Show()，分别用于保存数据和显示数据。其中保存数据仅使用了 setItem()方法，显示数据则根据索引遍历“键”，并根据“键”获取对应的“值”，使用了 key()方法和 getItem()方法。

12.1.5　存储 JSON 对象的数据

虽然使用 Web Storage 可以保存任意的键/值对数据，但是一些浏览器把数据限定为字符串类型，而且对于一些复杂结构的数据，似乎管理起来比较混乱，例如示例 12-4 中，如果要保存多个人的数据，就会变得不易于管理。

不过对于复杂结构的数据，可以使用现代浏览器都支持的 JSON 对象来处理，这也为我们提供了一种可行的解决方案。

1．序列化Json格式的数据

由于 Storage 是以字符串保存数据的，因此在保存 Json 格式的数据之前，需要把 Json 格式的数据转换为字符串，称为序列化。可以使用 JSON.stringify()序列化 Json 格式的数据为字符串数据。使用方法如下：

```
var stringData = JSON.stringify(jsonObject);
```

以上代码把 Json 格式的数据对象 jsonObject 序列化为字符串数据 stringData。

2．把数据反序列化为Json格式

如果把存储的 Storage 中的数据以 Json 格式对象的方式去访问，需要把字符串数据转换为 Json 格式的数据，称为反序列化。可以使用 JSON.parse()反序列化字符串数据为 Json 格式的数据。使用方法如下：

```
var jsonObject = JSON.parse(stringData);
```

以上代码把字符串数据 stringData 反序列化为 Json 格式的数据对象 jsonObject。

提示： 反序列化字符串为 Json 格式的数据，也可以使用 eval()函数，但 eval()函数是把任意的字符串转化为脚本，有很大的安全隐患。但是 JSON.parse()只反序列化 Json 格式的字符串数据，如果字符串数据不符合 Json 数据格式，则会产生错误，同时也减少了安全隐患。但是执行效率方面 eval()函数要快很多。

3．示例介绍

下面更改一下示例 12-4 中的脚本代码，存储 Json 格式的数据，每次单击“保存”按钮，均会增加一个键/值对，每个键的值都包含了一次的保存数据。

【示例 12-5】 使用 Storage 对象存储 Json 数据。

```
<script type="text/javascript">
var flag = 1;
 window.sessionStorage.clear();
//保存数据到 sessionStorage
function SaveStorage(frm){
```

```
    //使用表单数据建立 JSON 对象
    var jsonObject = new Object();
    jsonObject.name = frm.name.value;
    jsonObject.age = frm.age.value;
    jsonObject.email = frm.email.value;
    //序列化 JSON 对象为字符串数据
    var stringData = JSON.stringify(jsonObject);
    //存储字符串数据至 Storage
    var storage = window.sessionStorage;
    storage.setItem("key"+flag,stringData);
    //改变键标识
    flag++;
}
//遍历并显示 sessionStorage 中的数据
function Show(){
    var storage = window.sessionStorage;
    var result = "";
    for(var i=0;i<storage.length;i++){
        var key = storage.key(i);               /* 获取键 key */
        var stringData = storage.getItem(key);/* 通过键 key 获取值 stringData */
        var jsonObject = JSON.parse(stringData);/* 反序列化字符串为 JSON 对象 */
        //操作 JSON 对象，并显示存储的内容
        result += "姓名:"+ jsonObject.name+"; 年龄:"+ jsonObject.age+"; 邮
        件:"+ jsonObject.email+"<br>";
    }
    /* 在指定的地方显示获取的存储内容 */
    document.getElementById("formdata").innerHTML = result;
}
</script>
```

运行结果如图 12-5 所示。

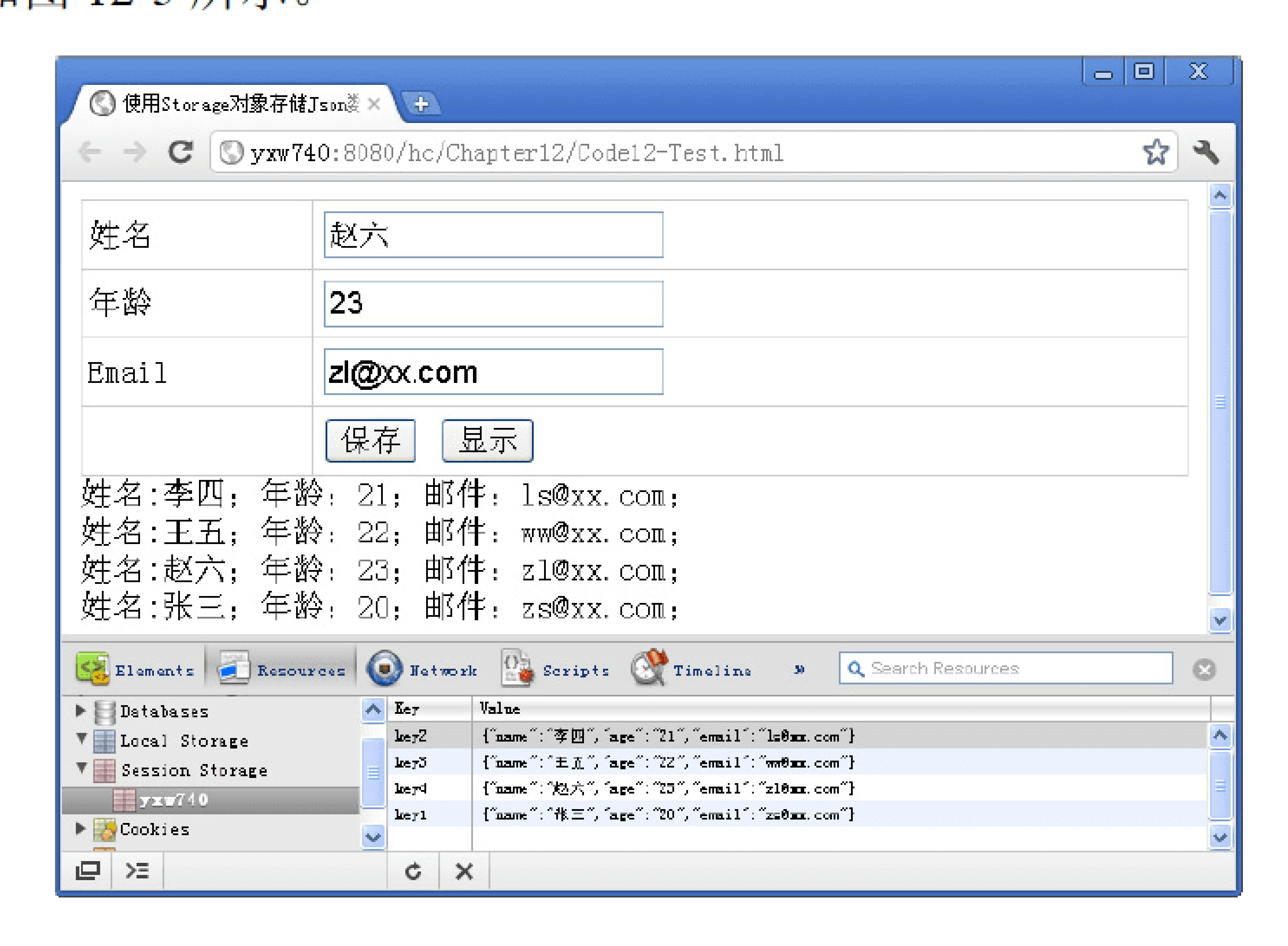


图 12-5　Storage 对象保存的页面内容

代码分析：在示例 12-5 中，保存数据时，先使用表单内容建立一个 JSON 对象，然后序列化 JSON 对象为字符串数据，保存至 Storage。显示数据时，会遍历所有存储的数据，并把读取的数据反序列化一个 JSON 对象，然后再对该对象进行操作。

显然，借助 JSON 对象来保存数据，可以把复杂的数据变得简单而有效。

12.1.6　Storage API 的事件

有时候，会存在多个网页或标签页同时访问存储的数据的情况。为保证修改的数据能够及时反馈到另一个页面，HTML 5 的 Web Storage 内建立一套事件通知机制，会在数据更新时触发。无论监听的窗口是否存储过该数据，只要与执行存储的窗口是同源的，都会触发 Web Storage 事件。

像下面这样，添加监听事件后，即可接收同源窗口的 Storage 事件：

```
window.addEventListener("storage",EventHandle,true);
```

storage 是添加的监听事件，本节主要介绍这个监听事件。只要是同源的 Storage 事件发生（包括 sessionStorage 和 localStorage），都能够因数据更新而触发事件。Storage 事件的接口如示例 12-6 所示。

【示例 12-6】 StorageEvent 事件接口。

```
interface StorageEvent : Event {
  readonly attribute DOMString key;
  readonly attribute DOMString? oldValue;
  readonly attribute DOMString? newValue;
  readonly attribute DOMString url;
  readonly attribute Storage? storageArea;
};
```

StorageEvent 对象在事件触发时，会传递给事件处理程序，它包含了与存储变化有关的所有必要的信息。

- key 属性：包含了存储中被更新或删除的键。
- oldValue 属性：包含了更新前键对应的数据。如果是新添加的数据，则 oldValue 属性值为 null。
- newValue 属性：包含了更新后的数据。如果是被删除的数据，则 newValue 属性值为 null。
- url 属性：指向 Storage 事件的发生源。
- storageArea 属性：该属性是一个引用，指向值发生改变的 localStorage 或 sessionStorage。这样，处理程序可以方便地查询到 Storage 中的当前值，或者基于其他的 Storage 执行其他操作。

12.1.7　实验室：在两个窗口中实现通信

当给其中一个页面添加 Storage 事件的时候，如果同源的另一个页面更改了存储的 Storage 数据，就会触发 Storage 事件，我们可以根据此事件获取最新的存储数据或执行其他处理。

下面一起看一个窗口通信示例。在窗口 1 中改变 Storage 数据，可在窗口 2 中获取数据并实时更新。

【示例 12-7】 窗口通信 Page1：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>窗口通信 Page1</title>
<script type="text/javascript">
function SaveStorage(frm){
     //保存数据到 localStorage
     window.localStorage.name=document.getElementById("name").value;
}
</script>
</head>
<body>
姓名<input type="text" name="name" id="name" />
<input type="button" value="保存" onclick="SaveStorage(this.form)" />
</body>
</html>
```

【示例 12-8】 窗口通信 Page2：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>窗口通信 Page2</title>
<script type="text/javascript">
function EventHandle (e){
     var storage = window.localStorage;
     var result = "";
     result+="<br />姓名:"+storage.name;
     result+="<br />key:"+e.key;
     result+="<br />oldValue:"+e.oldValue;
     result+="<br />newValue:"+e.newValue;
     result+="<br />url:"+e.url;
     result+="<br />storageArea:"+JSON.stringify(e.storageArea);
     /* 在指定的地方显示获取的存储内容 */
     document.getElementById("formdata").innerHTML = result;
}
//添加监听事件 Storage
window.addEventListener("storage",EventHandle,true);
</script>
</head>
<body>
<div id="formdata"></div>
</body>
</html>
```

运行结果如图 12-6 所示。

代码分析：在示例 12-7 中的 Page1 页面，会把输入表单里的姓名保存在 localStorage 中，而且可以随时更改存储的值。在示例 12-8 中的 Page2 页面，注册了 Storage 事件的监听，当存储在 localStorage 中的数据发生改变时，会触发 Page2 中的 Storage 事件，并执行 EventHandle()函数，把保存在 localStorage 里的数据显示出来。如图 12-6 所示，当修改 Page1 中的姓名时，Page2 中的内容会即时发生改变。

通过 Storage 实现的窗口通信，可以实现多个窗口的数据同步。如果是后台异步获取数据并更新 Storage 存储数据，同样可以利用此事件，完成数据的及时同步。

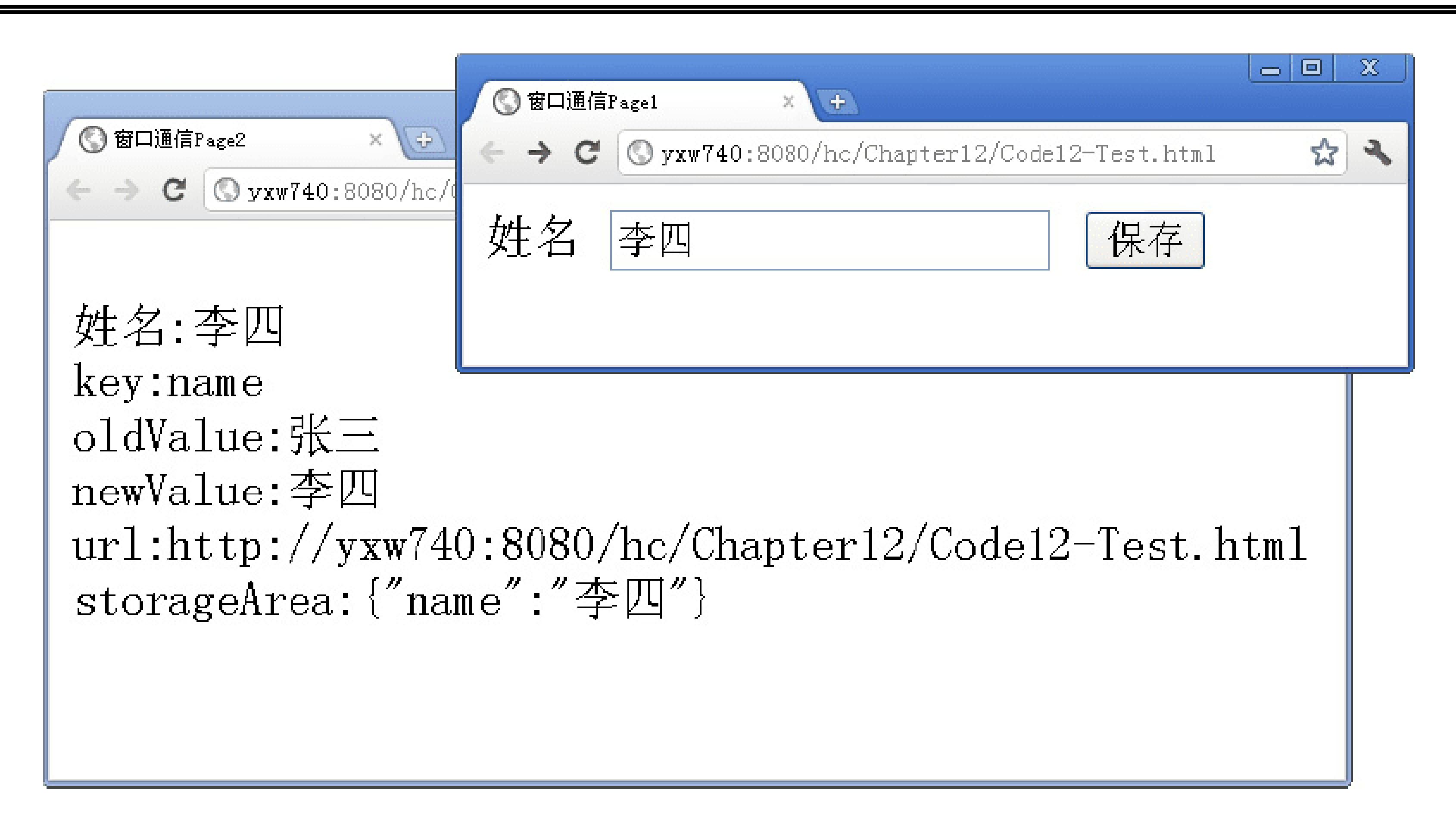


图 12-6　窗口通信

12.2　本地数据库——Web SQL Database

为了进一步加强客户端的存储能力，HTML 5 引入了本地数据库的概念。但 HTML 5 的数据库 API 的具体细节仍在完善，其中 Web SQL Database 就是数据库方案之一。实际上，Web SQL Database 并不包含在 HTML 5 规范之中，它是一个独立的规范，引入了使用 SQL 操作客户端数据库的 API。最新版本的 Chrome、Safari 和 Opera 浏览器都已经实现了它。

12.2.1　Web SQL Database 简介

Web SQL Database 规范使用的是 SQLite 数据库，它允许应用程序通过一个异步的 JavaScript 接口访问数据库。虽然 Web SQL 不属于 HTML 5 规范，而且 HTML 5 最终也不会选择它，但是对于移动领域是非常有用的，因为在任何情况下，SQL API 在数据库中的数据处理能力都是无法比拟的。

SQLite 是一款轻型的数据库，遵循 ACID 的关系型数据库管理系统。它的优势是嵌入式的，且占用资源非常低，只需要几百 KB 的内存就可以了。在跨平台方面，它能够支持 Windows/Linux 等主流操作系统；同时能够跟很多程序语言如 C#/PHP/Java/JavaScript 等结合；它包含 ODBC 接口；在处理速度方面也非常可观。

- Web SQL Database 规范中定义了如下 3 个核心的方法。
  - openDatabase()方法：使用现有的数据库或新建数据库来创建数据库对象。
  - transaction()方法：允许我们控制事务的提交或混滚。
  - executeSql()方法：用于执行真实的 SQL 查询。

提示： 在下面内容的介绍中，会涉及很多 SQL 语句，如果对 SQL 不熟悉，那么在继续学习本章内容之前，最好先学习一下 SQL 相关教程。

12.2.2　操作 Web SQL 数据库

1．打开数据库

openDatabase()方法可以打开一个已经存在的数据库，如果数据库不存在，它可以创建数据库。创建并打开数据库的语法如下：

```
var db = openDatabase("TestDB", "1.0","测试数据库",2*1024*1024, creation
Callback);
```

说明：该方式有 5 个必需的参数，第一个参数表示数据库名；第二个参数表示版本号；第三个参数表示数据库的描述；第四个参数表示数据库的大小；第五个参数表示创建回调函数。其中第五个参数是可选的。

2．创建数据表

transaction()方法可以进行事务处理；executeSql()方法可以执行 SQL 语句。可以同时使用这两个方法，在事务中处理 SQL 语句。创建数据表的方法如下：

```
db.transaction(function (tx){
     tx.executeSql('CREATE TABLE IF NOT EXISTS UserName (id unique, Name)');
     });
```

说明：使用 transaction()方法传递给回调函数的 tx 是一个 transaction 对象，然后使用 transaction 对象的 executeSql()方法，可以执行 SQL 语句。这里的 SQL 语句就是创建数据表的命令。

3．添加数据至数据库表

与创建数据表一样，也可以使用 transaction()方法和 executeSql()方法，仅仅是 SQL 语句不同。我们使用插入数据的 SQL 语句执行数据的插入操作。添加数据至数据库表的方法如下：

```
db.transaction(function (tx){
    tx.executeSql('INSERT INTO UserName (id, Name) VALUES (1, "张三")');
    tx.executeSql('INSERT INTO UserName (id, Name) VALUES (2, "李四")');
});
```

说明：两个包含 Insert INTO 命令的 SQL 语句，表示插入数据，将会在本地数据库 TestDB 中的 UserName 表中添加两条数据。

4．读取数据库中的数据

仍然使用 transaction()方法和 executeSql()方法，使用查询 SQL 语句，并在 executeSql()方法中，添加匿名的回调处理函数。

```
db.transaction(function (tx){
    tx.executeSql('SELECT * FROM UserName',[],function(tx,results){
        var len = results.rows.length;
        for (var i=0;i<len;i++){
            console.log(results.rows.item(i).Name);
        }
    }, null);
});
```

说明： executeSql()方法中执行了包含 Select 命令的 SQL 语句，表示查询，将从本地数据库 TestDB 中的 UserName 表中查询信息。查询出来的结果会传递给匿名的回调函数，我们可以在回调函数中处理查询的结果，如控制台输出结果。

12.2.3　实验室：基本的数据库操作示例

本节将使用 Web SQL 数据库实现数据的存储，并实现针对数据库的添加、更新、删除和查询等基本操作。

1．案例简介

设计一个简易的管理页面，包含一个填写姓名的表单，并有一个“查询”按钮，可以查询 Web SQL 数据库中的数据；“添加”按钮可以添加数据。如果单击列表中的“编辑”，则表单会针对该项进行编辑更新，实际效果如图 12-7 所示。



图 12-7　数据库操作示例

2．页面元素设计

设计的页面主要包含以下几个方面：用于填写姓名的输入框，能自动切换添加/更新的按钮，一个“查询”按钮。表单下面是显示列表的区域。页面按钮的 click 事件处理函数会在后面实现。

【示例 12-9】 本地数据库操作页面。

```
<!DOCTYPE HTML>
<html>
<head>
<title>本地数据库访问示例</title>
<style type="text/css">
...... <!-- 省略样式表 -->
</style>
</head>
<body>
<form>
    <input id="id" name="id" type="hidden" />
    <input id="name" name="name" type="text" placeholder="请输入姓名" />
    <input type="button" id="Submit" name="Submit" value="添加" onclick="
    Insert()" />
    <input type="button" value="查询" onclick="Query()" />
</form>
<ul id="msg" name="msg">
</ul>
</body>
</html>
```

3．初始化页面的脚本

页面加载时初始化，做两个处理操作。首先打开数据库，如果数据库不存在，则创建数据库；其次是确保数据表存在，如果不存在，则创建数据表。

```
<script type="text/javascript">
/* 打开数据库，如果不存在则创建 */
var db = openDatabase("TestDB", "1.0","测试数据库",2*1024*1024);
/* 创建/打开数据表，如果存在则不创建 */
db.transaction(function (tx){
    tx.executeSql('CREATE TABLE IF NOT EXISTS UserName (id unique, Name)');
    });
</script>
```

4．实现查询功能

查询按钮的 click 事件处理函数 Query()，可根据关键词来查询数据库中的数据，其中关键词来源于表单中的姓名输入框。

另外，初始化表单处理函数，主要应用于两个方面：一是编辑信息时，把其中的内容初始化至表单中，“添加”按钮变为“更新”按钮；二是保存数据和删除数据后，初始化表单内容为空，并把按钮初始化为“添加”。

```
<script type="text/javascript">
/* 查询数据 */
function Query(){
    var name = document.getElementById("name");
    db.transaction(function (tx){
        tx.executeSql('SELECT * FROM UserName where Name like"%'+ name.
        value +'%" ORDER BY id DESC',[],function(tx,results){
           var len = results.rows.length;
           var msg="";
           for(var i=0;i<len;i++){
               msg += "<li>&middot; ";
               msg += "<span>" + results.rows.item(i).Name + "</span>";
```

```
                    msg += " <a href='###' onclick=\"SetForm('"+ results.
                    rows.item(i).id +"','"+ results.rows.item(i).Name +"')
                    \">编辑</a>";
                    msg += " <a href='###' onclick='Delete("+ results.rows.
                    item(i).id +")'>删除</a>";
                    msg += "</li>";
                }
                document.getElementById("msg").innerHTML = msg;
            }, null);
        });
}
/* 初始化表单 */
function SetForm(id,name){
    if(id){
        document.getElementById("id").value=id;
        document.getElementById("name").value=name;
        document.getElementById("Submit").onclick=function(){Update();}
        document.getElementById("Submit").value="更新";
    }else{
        document.getElementById("id").value="";
        document.getElementById("name").value="";
        document.getElementById("Submit").onclick=function(){Insert();}
        document.getElementById("Submit").value="添加";
    }
}
</script>
```

5．实现添加、修改和删除功能

由于插入到数据库中的数据必须包含编号（id）和姓名（name）两方面的信息，添加处理函数 Insert()只能从页面获取姓名，而编号则需要查询数据库以获取可用的最小值。最后才去保存数据，并在保存数据的 executeSql()方法中，添加保存成功的匿名的回调函数：初始化表单和显示所有信息。

更新处理函数 Update()，可以从表单获取两个信息编号（id）和姓名（name），所以可以直接操作数据库更新信息，并添加同样的保存成功的匿名的回调函数。

删除处理函数 Delete(id)，传递了一个编号（id），通过编号可以直接删除数据表中的记录。

所有的操作实现如下：

```
<script type="text/javascript">
/* 添加数据 */
function Insert(){
     var name = document.getElementById("name");
     if(name.value == "")return; /* 没有值，则不处理 */
     var maxid;
     /* 获取可用的最小 id 值 */
     db.transaction(function (tx){
          tx.executeSql('SELECT id FROM UserName ORDER BY id DESC',[],
          function(tx,result){
             if(result.rows.length){
                   maxid = parseInt(result.rows.item(0).id) + 1;
             }else{
                   maxid = 1;
             }
```

```
            }, null);
        });
        /* 添加一条数据，并更新显示 */
        db.transaction(function(tx){
            tx.executeSql('INSERT INTO UserName (id, Name) VALUES ('+ maxid +',
            "'+ name.value +'")',[],function(tx,result){
                SetForm();
                Query();
            });
        });
}
/* 更新数据 */
function Update(){
    db.transaction(function(tx){
        var id = document.getElementById("id");
        var name = document.getElementById("name");
        console.log(name.value);
        tx.executeSql('Update UserName Set Name = "'+ name.value +'" where
        id='+ id.value,[],function(tx,result){
            SetForm();
            Query();
        });
    });
}
/* 删除数据 */
function Delete(id){
    db.transaction(function(tx){
        tx.executeSql('Delete From UserName where id='+ id,[],function
        (tx,result){
            SetForm();
            Query();
        });
    });
}
</script>
```

至此，与数据有关的基本操作已经完成，我们可以在运行的页面上实现数据的添加、修改、删除和查询，如图 12-7 所示。

12.3　小　　结

本章主要讲解了 HTML 5 的本地存储的概念，包括 HTML 5 的 Web Storage 和独立规范的 Web SQL 数据库。其中重点讲解了 Web Storage 接口的方法和事件，并介绍如何利用 Web Storage 事件实现窗口间的数据通信。另外还重点讲解了 JSON 对象，以扩展 Storage 对象存储的复杂性。本章中比较难的部分是对 Storage 编程接口的理解，以及与 sessionStorage 和 localStorage 的关系。Web SQL 数据库对于很少接触数据库的前端工程师来说，是一个大的挑战。下一章将介绍 HTML 5 著名的离线应用。

12.4　习　　题

【习题 1】Web Storage 本地存储包括哪两个方面？

【习题 2】JSON 对象可以协助 Web Storage 保存复杂结构的 Json 格式的数据，它是如何对 Json 格式的数据进行序列化和反序列化的？

【习题 3】一旦 Web Storage 本地存储的数据发生变化，即可触发 Storage 事件。请编写两个页面：在第一个页面中更新 sessionStorage 或 localStorage，然后在第二个页面上监听这些数据的变化。

【习题 4】Web SQL Database 可以在本地存储更多更复杂的数据，目前 Web SQL Database 规范中定义了哪三个核心的方法?

第 13 章　别开生面的离线应用

随着越来越多的应用移植到了 Web 上，人们对 Web 应用的依赖逐渐增强，对 Web 应用的要求也越来越高。但是 Web 应用通常都有一个致命的缺陷，就是如果用户不能连接网络或网络不畅通，就无法使用 Web 应用程序。为了适应网络环境不佳或减少对网络的占用，HTML 5 综合 Web 应用和桌面应用的优势，提供了在本地缓存应用程序的 API，也叫 Web 的离线应用 API。本章将详细讲述 HTML 5 的离线应用接口，以及与之相关的本地缓存文件清单 manifest。

13.1　Web 离线应用缓存

HTML 5 新增的离线的 Web 应用，代表 Web 领域发展的新方向，即便是在没有网络的情况下，仍然可以使用网络资源。而离线应用则是通过离线应用缓存实现的。

1．新增的离线应用缓存

传统的 Web 应用程序常常会遇到一个很大的问题，就是网络不好时，就无法使用网络上提供的应用程序。HTML 5 借鉴了桌面应用程序的特征，引入了离线应用缓存。

Web 应用程序可通过浏览器的离线应用缓存功能，提前把与应用相关的资源文件缓存到本地，当断开网络或网络环境不佳时，用户仍然可以继续浏览未浏览完成的内容。

离线应用缓存功能的另一个好处是可以永久地缓存静态的内容，并且没有缓存过期的限制，这样即便在网络畅通的情况下，仍然会使用本地缓存的文件，避免了与服务器的过多交互。这样，一方面节省了网络资源，另一方面也能减轻服务器的访问压力。

离线应用缓存需要以一种方式来指明应用程序离线时所需要的资源文件。这样，浏览器才能在在线状态时，把这些资源文件缓存到本地。此后，当用户离线访问应用程序时，这些资源文件会自动加载。在 HTML 5 中可通过一个缓存清单的文件 manifest 指明需要缓存的资源，并且支持自动和手动两种缓存更新方式。

此外，HTML 5 还提供了在线状态的检测，应用程序需要检测网络是否畅通，这样才能够针对在线和离线的状态，做出相应的处理。HTML 5 提供了两种在线状态的检测方式。

2．离线应用缓存与传统页面缓存的区别

离线应用缓存与传统的页面缓存有着巨大的差别。

首先，离线应用缓存是为 Web 应用程序服务的，是通过一个缓存清单来缓存 Web 应用程序所需要的资源文件的；而传统的网页缓存，仅服务于单个页面，当浏览到该页面时，才会产生页面缓存。

其次，离线应用缓存是安全可靠的。页面缓存是由浏览器自动完成的，无法确定用户在本地缓存了哪些页面。而离线应用，需要开发人员来指定这些资源缓存的内容，是通过服务器端控制的，这样，一方面我们可以保证用户可以正确地使用 Web 应用程序；另一方面，Web 应用程序更新也非常及时。

使用离线应用缓存，可以开发出更加强大的 Web 应用程序，使用起来就如同本机的应用程序一样。

3．离线应用缓存与本地数据存储的区别

离线应用缓存与本地数据存储是两个不同的概念。

离线应用缓存是把应用程序所需要的资源文件，从服务器端缓存到本地的一种缓存机制。

而本地数据存储是在本地存储数据，仅仅是客户端的行为，不会与服务器发生任何交互行为，是为了满足不同的存储需求而提供的一种数据存储机制。

13.2　缓存清单文件 manifest

为了使用户能够在离线状态下使用 Web 应用程序，需要开发者在服务器端提供一个缓存清单 manifest 文件。此清单中，列出了离线应用程序需要的所有资源文件。访问服务器时，浏览器会把这些资源文件缓存到本地。

1．manifest文件的格式

下面通过一个 manifest 文件清单示例进行详细介绍。

【示例 13-1】 manifest 清单文件：

```
CACHE MANIFEST
# 文件的开头必须是 CACHE MANIFEST
# 可以在这里设置文件的版本号
# version1.1
CACHE:
css/style.css
script.js
NETWORK:
images/pictrue.jpg
*
FALLBACK:
css/style2.css css/style.css
CACHE:
other.html
```

则浏览器会在本地缓存 manifest 文件所列出的资源文件，如图 13-1 中的矩形框区域所示。

关于示例 13-1 所描述的 manifest 清单文件，说明如下。

- 文件内容的第一行必须是“CACHE MANIFEST”，用于把文件的作用告诉浏览器，浏览器就会把本文件所列出的文件资源进行客户端缓存。
- 文件中的注释需要另起一行，注释行是以符号“#”开始的行。如果“#”出现在

Url 中，会被认为是 Url 的一部分。

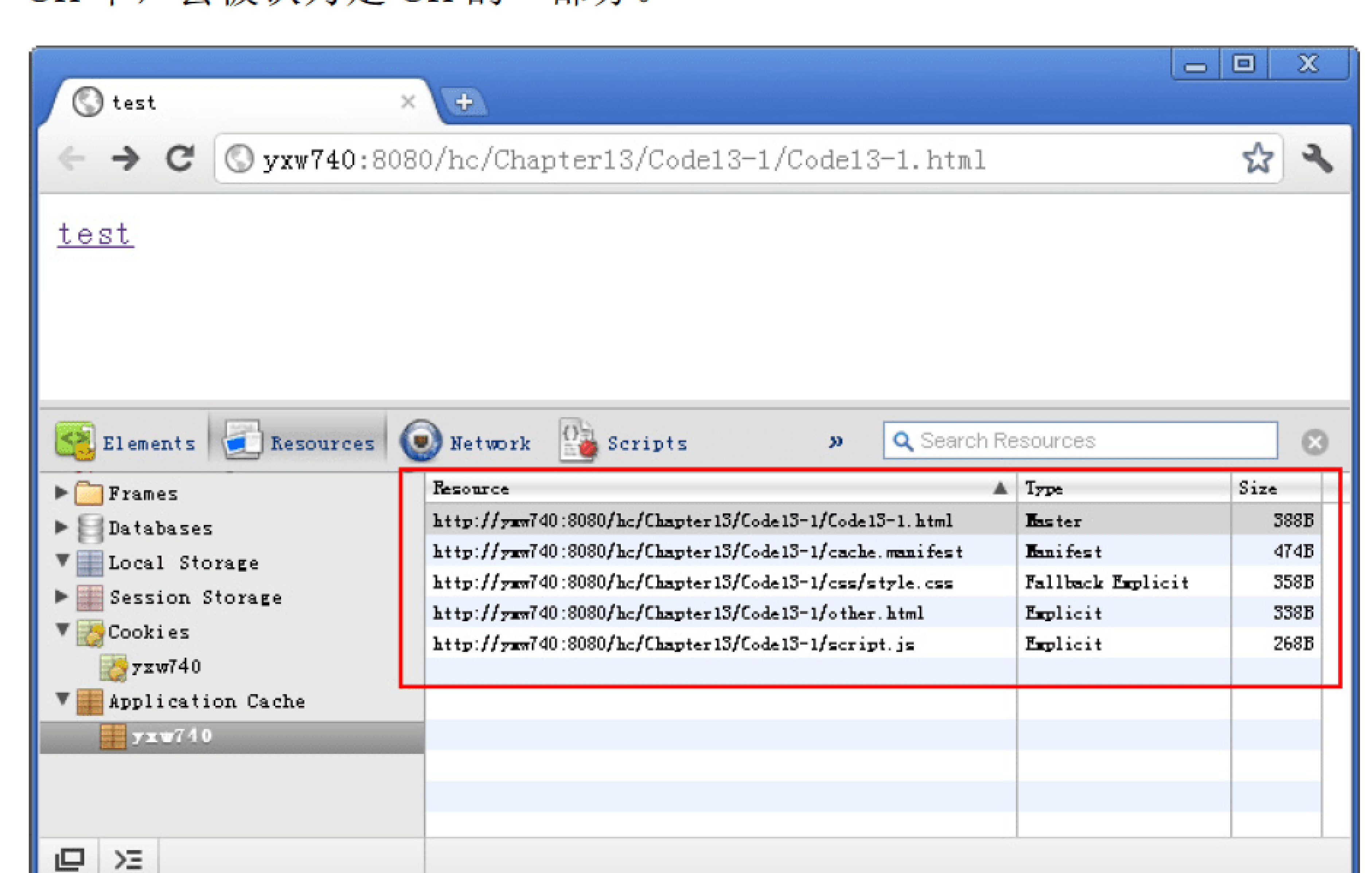


图 13-1　缓存在本地的资源文件

- 最好给 manifest 清单文件加上一个版本注释，如“version1.1”。该注释并没有什么意义，主要是提醒浏览器更新缓存文件。
- 对于指定的缓存文件，可以是绝对路径，也可以是相对路径。本示例使用的是相对路径。
- 文件中有三个关键字用于资源的分类，分别是 CACHE、NETWORK 和 FALLBACK。
- CACHE 类别中指定的文件是要被缓存到本地的资源文件。
- NETWORK 类别指定的文件是不会缓存到本地的资源文件，只有在网络畅通的情况下才能被访问。
- FALLBACK 类别中，每行指定两个资源文件。第一个文件为能够在线访问时使用的资源文件；第二个文件是不能在线访问时的资源文件。
- 三个类别中，每个类别都是可选的。但是如果 manifest 文件没有指定任何类别就列出资源文件，那么就是 CACHE 类别的资源文件。

2. 使用manifest文件的方法

页面元素 html 有一个特性是 manifest，用于指定 manifest 缓存清单文件，浏览器可通过该清单文件缓存 Web 应用程序所需要的资源文件。使用方法如下：

```
<!DOCTYPE HTML>
<html manifest="cache.manifest">
...
</html>
```

manifest 特性指定了一个缓存清单文件 cache.manifest。浏览器通过识别该清单文件的第一行来确定文件的类型是否为用于离线应用缓存的清单文件。

注意：指定了 manifest 缓存清单文件的页面，不需要再列入 manifest 文件中，浏览器会默认对该页面进行缓存。

3. 服务器端的配置

manifest 文件的 MIME 类型是 text/cache-manifest，在使用之前，要确保服务器能够支持该文件。为此，我们需要在服务器端做一些配置。下面针对各个主流服务器分别给出其配置方法。

IIS 服务器中的配置需要如下步骤。

- 右键单击需要添加类型的网站，弹出对话框。
- 选择“HTTP 头”标签。
- 在 MIME 映射下，单击“文件类型”按钮，弹出 MIME 类型对话框。
- 在关联扩展名的文本框中输入“manifest”，在内容类型文本框中，输入“text/cache-manifest”，然后单击“确定”按钮即可。

Apache 服务器中的配置方法是：找到{apache_home}/conf/mime.types 文件，并在文件中添加如下所示的一行代码即可：

```
text/cache-manifest manifest
```

Tomcat 服务器中的配置方法是：找到{tomcat_home}/conf/webxml 文件，并在文件中添加如下配置节即可：

```
<mime-mapping>
    <extension>manifest</extension>
    <mime-type>text/cache-manifest</mime-type>
</mime-mapping>
```

Python 标准库中的 SimpleHTTPServer 的配置方法是：找到{PYTHON_HOME}/Lib/mimetypes.py 文件，并在文件中添加如下一行代码即可：

```
'.manifest':'text/cache-manifest manifest',
```

有了服务器的支持，才能够正确地使用离线应用缓存的功能。

13.3　检测浏览器的网络状态

离线的 Web 应用程序在离线状态和在线状态下，有不同的行为模式，HTML 5 引入了一些新的事件，用于检测网络能否正常连接。通常是使用 window.navigator 对象的 onLine 属性来检测。

window.navigator 对象的 onLine 属性是一个标明了浏览器是否处于在线状态的布尔值。当 onLine 属性值为 false 时，Web 应用程序不会尝试进行网络连接；当属性值为 true 时，会尝试进行网络连接，但不一定能确保访问到相应的服务器。下面通过示例来介绍网络状态的检测方法。

【示例 13-2】 检测在线状态。

```
<script type="text/javascript">
//检测在线状态
function TestingNetwork(){
    if(window.navigator.onLine){
        console.log("在线状态");
    }else{
        console.log("离线状态");
    }
}
//添加在线状态监听器
window.addEventListener("online",function(e){
    console.log("在线状态");
},true);
//添加离线状态监听器
window.addEventListener("offline",function(e){
    console.log("离线状态");
},true);
</script>
```

说明： 在示例 13-2 中，介绍了两种检测网络状态的方法：一种是通过判断 window.navigator.onLine 属性，来确定网络的状态；另一种是通过添加事件监听器的方法，来监听网络的状态。

13.4　应用缓存接口 applicationCache

HTML 5 提供了一系列操作应用缓存的接口，均包含在新增的 window.applicationCache 对象中，可触发一系列与缓存相关的事件。

1．检测浏览器支持情况

使用 Web 的离线应用，最好先检查浏览器是否支持它。这也是使用 HTML 5 新增接口的好习惯，必要的支持性检测可以防止不必要的错误发生。

【示例 13-3】 检测浏览器是否支持离线应用。

```
<script type="text/javascript">
//检测浏览器是否支持离线应用
if(window.applicationCache){
    console.log("浏览器支持离线应用");
}else{
    console.log("浏览器不支持离线应用");
}
</script>
```

2．applicationCache接口

离线应用缓存的接口包括基本的属性、方法和事件，并且定义了离线应用的 6 种状态。

【示例 13-4】 applicationCache 接口。

```
interface ApplicationCache {
```

```
    //更新状态
    const unsigned short UNCACHED = 0;
    const unsigned short IDLE = 1;
    const unsigned short CHECKING = 2;
    const unsigned short DOWNLOADING = 3;
    const unsigned short UPDATEREADY = 4;
    const unsigned short OBSOLETE = 5;
    //只读属性
    readonly attribute unsigned short status;
    //更新方法
    void update();
    void swapCache();
    //事件
    attribute Function onchecking;
    attribute Function onerror;
    attribute Function onnoupdate;
    attribute Function ondownloading;
    attribute Function onprogress;
    attribute Function onupdateready;
    attribute Function oncached;
    attribute Function onobsolete;
};
ApplicationCache implements EventTarget;
```

下面我们就对 applicationCache 接口做详细讲解。

3．接口的属性status

应用缓存接口有一个状态属性 status，表示应用缓存的状态，并且该缓存是只读的。HTML 5 为该状态定义了 6 个数值常量，详情如表 13.1 所示。

表 13.1　应用缓存的 6 种状态

缓存状态	数值	说明	缓存状态	数值	说明
UNCACHED	0	未缓存	DOWNLOADING	3	下载中
IDLE	1	空闲	UPDATEREADY	4	更新就绪
CHECKING	2	检查中	OBSOLETE	5	过期

一般的网页都没有使用离线应用的功能，这些页面的应用缓存状态就是未缓存状态 UNCACHED。空闲状态 IDLE 是离线应用的典型状态，说明应用程序所需要的所有资源文件都已经被缓存到本地，不需要再更新。如果曾经有应用缓存，却发现 manifest 文件丢失，则缓存会进入过期状态 OBSOLETE。

4．接口的方法

应用缓存接口包含两个方法：update()和 swapCache()。

- update()方法：用于请求浏览器更新缓存。当方法被调用时，浏览器会检测 manifest 清单文件，如果有更新，就会重新下载缓存的资源文件。该方法执行完成后，应用缓存状态变成 UPDATEREADY，同时会触发 updateready 事件。
- swapCache()方法：用于手工执行本地缓存的更新。只能在 applicationCache 的 updateready 事件被触发时调用。而 updateready 事件只有在服务器端的 manifest 文件被更新，且把文件内的所有资源文件下载到本地后触发，而这一过程是 update()

方法所经历的。swapCache()方法只是让本地的应用缓存能够即时更新，而不是等到下一次刷新页面时更新。

5. 接口的事件

HTML 5 为应用缓存提供了 8 个事件用于编程开发，详情如表 13.2 所示。

表 13.2　应用缓存的事件

事　件	说　明
checking	检测应用缓存时触发。这始终是整个应用缓存过程中的第一个事件
error	发生任何错误时触发
noupdate	缓存的资源文件清单没有改变时触发
downloading	浏览器发现更新并获取时，或者下载清单中第一次列入的资源时触发
progress	浏览器正在下载资源文件时触发
updateready	清单列表中所有资源文件都更新完成时触发
cached	清单列表中所有资源文件都下载完成时触发
obsolete	找不到应用缓存清单manifest文件时触发

6. 测试接口事件的发生顺序

下面通过一个示例来了解应用缓存接口事件的发生顺序。

【示例 13-5】 测试接口事件的发生顺序。

```
<script type="text/javascript">
window.applicationCache.onchecking=function(){
    console.log("检查应用缓存更新");
}
window.applicationCache.onnoupdate=function(){
    console.log("没有需要更新的应用缓存");
}
window.applicationCache.onupdateready=function(){
    console.log("应用缓存更新就绪");
}
window.applicationCache.onobsolete=function(){
    console.log("应用缓存过时/过期");
}
window.applicationCache.oncached=function(){
    console.log("应用缓存下载完毕");
}
window.applicationCache.onerror=function(){
    console.log("应用缓存出现错误");
}
window.applicationCache.onprogress=function(){
    console.log("应用缓存下载中");
}
window.applicationCache.ondownloading=function(){
    console.log("应用缓存准备下载");
}
</script>
```

通过浏览器控制台查看输出结果如图 13-2 所示。

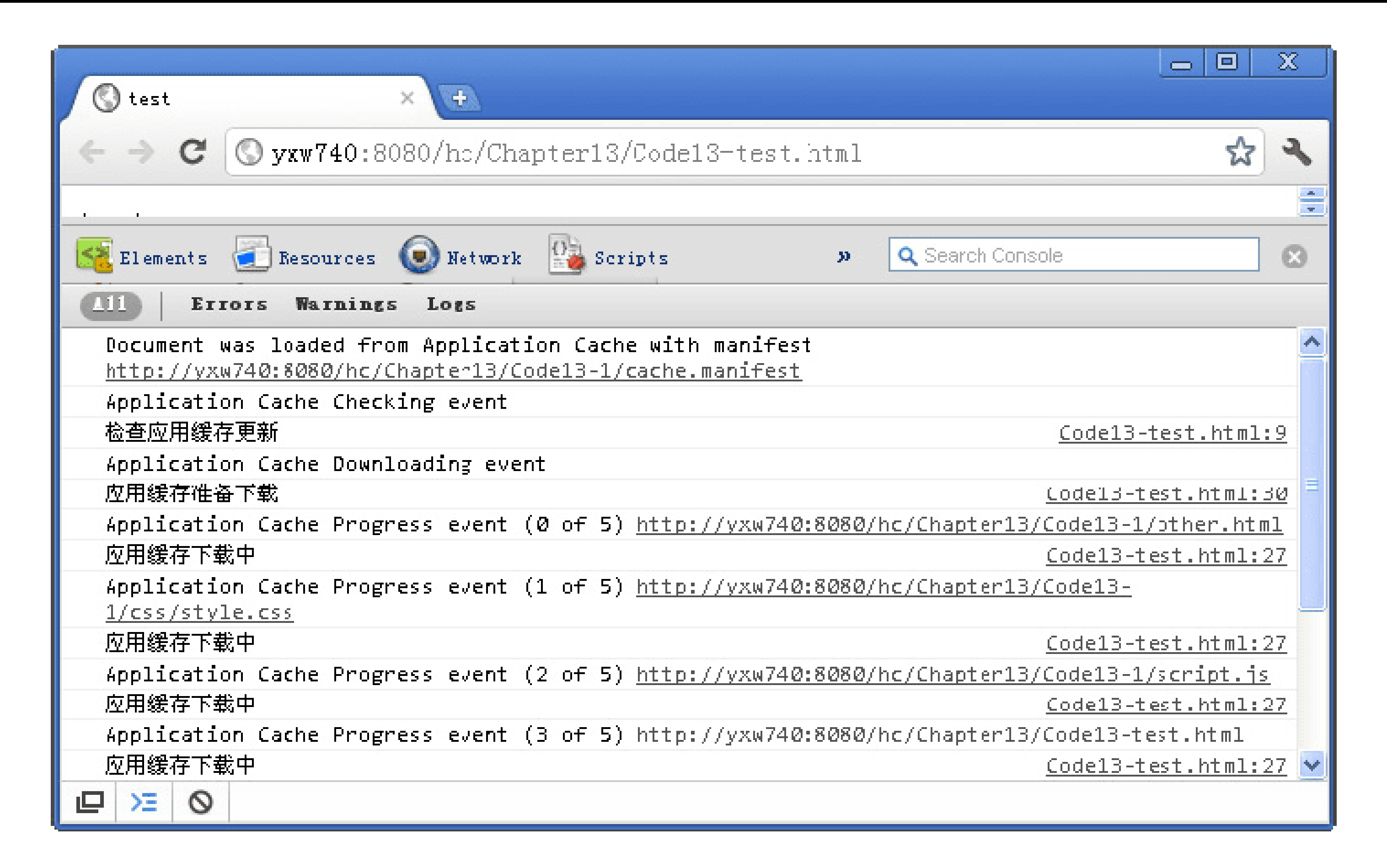


图 13-2　控制台输出结果

13.5　实验室：图片画廊的离线应用

在网络中，图片预览是一种最常用的 Web 应用。其中会涉及很多图片，而且对清晰的文件进行访问时，会占据很大的网络带宽，以至于每次加载该页面时，都会有漫长的等待过程。本节就介绍如何使用应用缓存来加快页面的加载速度，以及如何应对网络环境不佳的情况。

1．案例简介

本节的案例主要是对 6.1.9 节中的图片画廊案例进行改进，使之即便在离线的情况下也能使用，同时在后续的访问中，页面内容无须再次从服务器加载，而是直接读取本地应用缓存，从而提高访问速度，以获得更佳的用户体验。

在本节介绍的案例中，对于离线应用所需要的服务器配置不再讲述仅讲述离线资源文件清单的设计和页面操作 applicationCache 的功能。

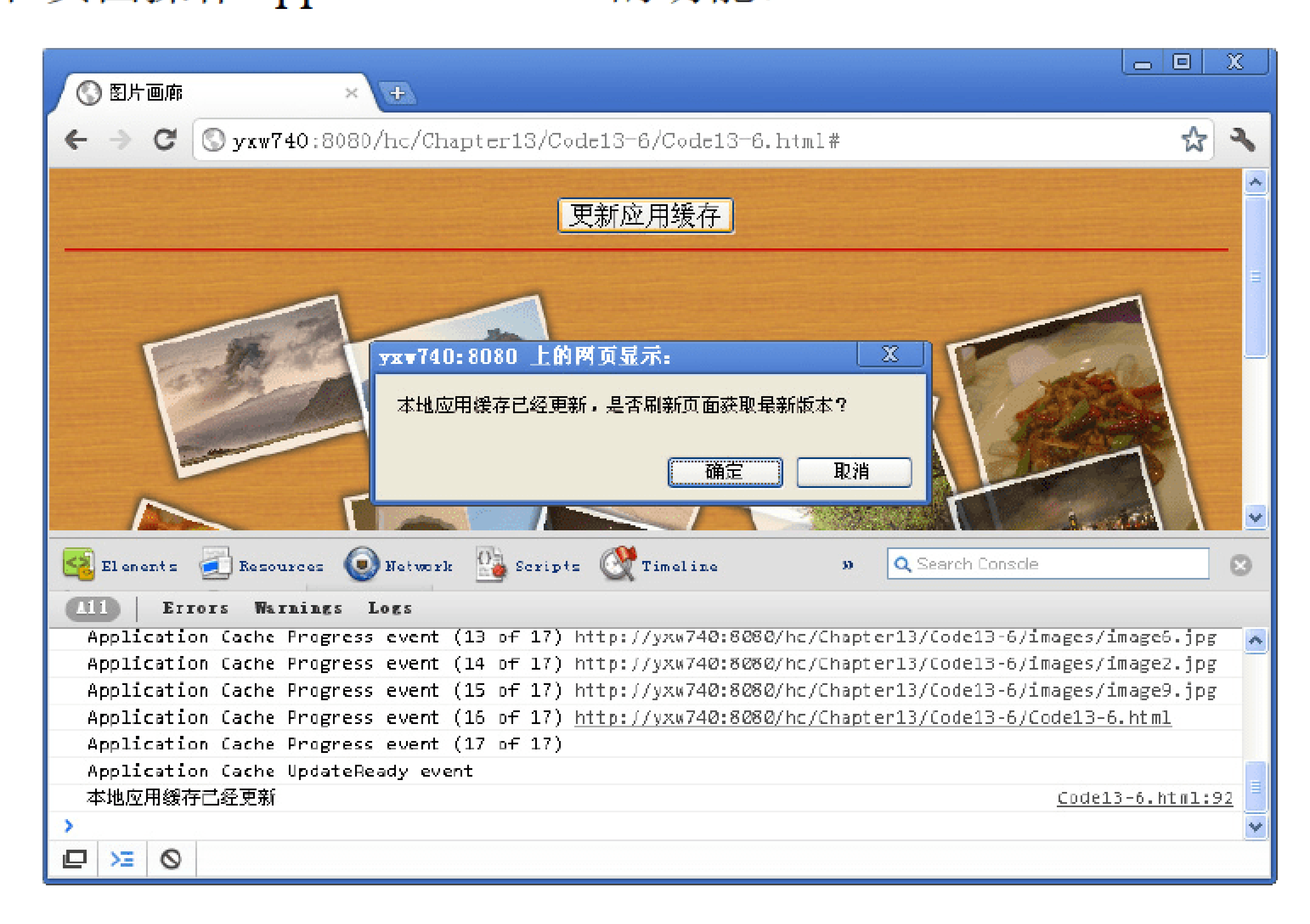


图 13-3　改进的图片画廊

2．设计网页元素

在网页里以列表的形式添加 15 张图片。由于图片文件过多，我们增加离线应用缓存功能，在 HTML 标签内添加 manifest 特性，并指定缓存清单文件 application.manifest。

【示例 13-6】 图片画廊的离线应用。

```
<!DOCTYPEHTML>
<html manifest="application.manifest">
<head>
<meta charset="utf-8">
<title>图片画廊</title>
<style type="text/css">
<!-- 样式表省略 -->
</style>
</head>
<body>
<div><input type="button" id="update" value="更新应用缓存" /></div>
<ul id="gallery">
  <li><a href="#" title="图片 1"><img src="images/image1.jpg" alt="图片 1"
/></a></li>
  <li><a href="#" title="图片 2"><img src="images/image2.jpg" alt="图片 2"
/></a></li>
  <li><a href="#" title="图片 3"><img src="images/image3.jpg" alt="图片 3"
/></a></li>
  … 省略了相似的代码
  <li><a href="#" title="图片 15"><img src="images/image15.jpg" alt="图片
15" /></a></li>
</ul>
</body>
</html>
```

3．设计application.manifest文件

按照以下方式设计 application.manifest 文件，文件中列出了图片画廊应用所需要的所有资源文件。

```
CACHE MANIFEST
# 文件的开头必须是 CACHE MANIFEST
# 可以在这里设置文件的版本号
# version1.0
CACHE:
images/image1.jpg
images/image2.jpg
images/image3.jpg
… 省略了相似的代码
images/image15.jpg
images/bark.jpg
```

这样，当用户浏览该页面时，页面会先检查是否需要更新资源文件，如果 manifest 文件没有发生过任何改变，则不会再从服务器上下载网页内容。如果 manifest 文件发生过改变，则重新下载离线资源文件以更新本地应用缓存。

如果资源文件发生了改变，而 manifest 清单文件没有改变，则浏览器不会更新该资源文件，但如果更改 manifest 文件的版本，则被认为 manifest 文件发生了改变，所有列出的

资源文件才会更新本地缓存。这也是设置 manifest 文件版本的好处。

以下介绍如何使用脚本对页面进行处理。

4. 设计手动更新应用缓存的脚本

下面为页面中的“更新应用缓存”的按钮设计脚本处理事件。

首先在页面加载完成后，检查浏览器是否支持离线应用缓存。如果支持离线应用缓存，则为该按钮事件添加缓存处理函数 UpdateCache()，否则就提示不兼容。

接下来的处理函数 UpdateCache()先检测服务器端的 manifest 文件是否更新，如果有更新则重新下载 manifest 文件中的资源文件；否则不处理。

最后，在函数 UpdateCache()中添加 updateready 事件处理，并在其中通过接口方法 swapCache()来及时更新新的应用程序缓存。

在示例 13-6 中添加脚本处理代码如下：

```
<script type="text/javascript">
function UpdateCache(){
    //检测是否有更新，有更新则下载更新的资源文件
    window.applicationCache.update();
    //应用缓存更新完成后的事件处理
    window.applicationCache.onupdateready=function(){
        console.log("本地应用缓存已经更新");
        if(confirm("本地应用缓存已经更新，是否刷新页面获取最新版本？")){
            //更新本地应用缓存
            window.applicationCache.swapCache();
            //重载页面
            window.location.reload();
        }
    }
}
window.onload=function(e){
    //检查浏览器是否支持离线应用
    if(window.applicationCache){
        document.getElementById("update").onclick = UpdateCache;
    }else{
        document.getElementById("update").onclick = function(){
            alert("浏览器不支持离线应用缓存");
        };
    }
}
</script>
```

至此，改进的图片画廊的离线应用已经完成。只要用户访问过该图片画廊应用，当再次访问图片画廊时，就不再受网络的限制，仍然可以正常地使用图片画廊。即便是网络畅通，该应用也不会首先从网络服务器上下载图片资源，在很大程度上减少频繁刷新带来的对网络带宽的占用。

13.6　小　　结

本章主要讲解了 HTML 5 的离线应用的概念。重点讲解了缓存清单 manifest 文件和应

用缓存接口 applicationCache 的属性、方法和事件。本章的难点在于 manifest 文件的使用方法，其中涉及各种服务器的配置；另一个难点就是对应用缓存接口的理解，其中包括 update()方法和 swapCache()方法的使用区别，以及众多的事件处理。这些对于前端开发人员来说，都是新的知识、新的挑战。

下一章将介绍跨文档的消息传输。

13.7　习　　题

【习题 1】请解释一下离线应用缓存与本地数据存储的区别。

【习题 2】请尝试编写一个缓存清单文件 manifest。

【习题 3】applicationCache 对象提供了一系列的事件以跟踪资源的下载状况。当应用缓存更新完成时，应触发哪个事件（单选）：

A．checking　　B．cached　　C．progress

D．downloading　　E．updateready　　F．obsolete

第 14 章　安全的跨源通信

随着 Web 应用越来越丰富，与来自不同站点及其页面进行通信显得非常重要。由于浏览器都遵循同源策略，不同站点的网页如果需要信息交换就变得棘手和复杂。也正是基于这种需求，HTML 5 提供了两个重要的通信模块：跨文档消息传输（Cross Document Messaging）和跨源异步请求。利用这两个模块，可以在不同的 Web 应用域之间进行非常安全的通信，其中跨文档消息传输用于不同域页面之间的脚本通信，跨源异步请求用于请求不同域的服务器资源。本章就详细介绍这两个方面。

14.1　跨文档消息传输

跨文档消息传输用于实现在不同的框架之间、窗口之间和标签之间的跨源通信。HTML 5 提供了完善的接口规范，在跨源通信的同时也能够保证其足够安全。

14.1.1　跨文档消息传输的实现

HTML 5 新增了网页之间的脚本通信功能，实现了最基本的相互发送和接收信息。在实现这种通信机制前，需要保证网页之间能够获取到对方窗口对象的实例。可以实现同源网页之间的通信，也可以实现跨源网页之间的 通信。

如图 14-1 所示的跨文档消息传输示意图，服务器 A 的页面 a 和服务器 B 的页面 b 会有不同的域名或端口号。页面 a 和页面 b 之间的消息通信就是本节讲的跨文档消息传输，一切的通信仅限于浏览器客户端。

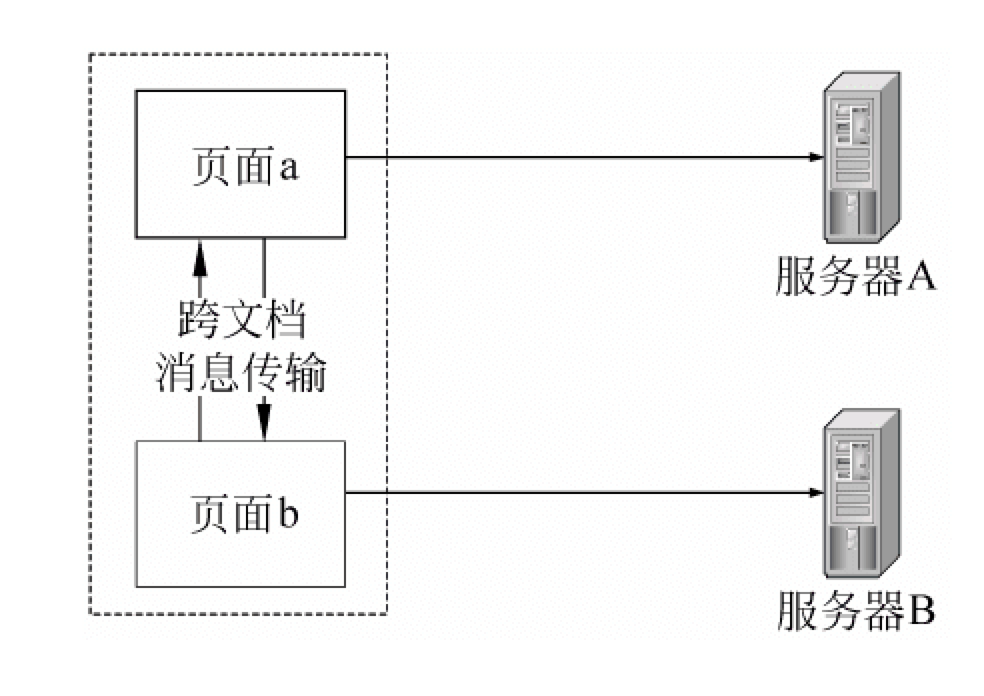


图 14-1　跨文档消息传输示意图

HTML 5 把 postMessage 接口定义为发送消息的标准方式。可使用 window 对象的 postMessage 方法向其他窗口发送消息。使用方法如下：

```
refWindow.postMessage(message,targetOrigin);
```

说明： refWindow 表示窗口对象的引用，我们要把消息传递到该窗口中；message 表示发送的消息，可以是一般的文本，也可以是转化成文本的对象；targetOrigin 表示接收消息窗口所属的源的 URL，例如 http://localhost:8000/。

为了使接收消息的窗口能够接收到消息，还需要给该窗口增加 message 事件，以监听其他窗口发送过来的消息。使用方法如下：

```
window.addEventListener("message", messageListener, false);
function messageListener(e){
    console.log(e.origin);
    console.log(e.data);
    console.log(e.source);
}
```

说明： message 为事件名称，触发该事件后，会调用一个回调函数 messageListener，以处理接收过来的消息。消息事件包含一个消息来源（origin）、发送的数据（data）和源窗体对象实例（resorce）。我们可以通过源信息，判断消息来源是否合法。

在此之前，网页之间的通信会通过脚本直接调用实现，即一个运行的页面会尝试调用另一个页面的数据。但是如果页面来源于不同的站点，会受到浏览器同源策略的限制。由于 postMessage 接口在同源页面和跨源页面中都可以使用，建议使用该接口实现跨文档的通信，以避免直接调用产生的不一致性。

14.1.2　Web 源安全

Web 的源安全是 Web 应用的基础安全，浏览器基本都遵守同源策略。HTML 5 打破了这种同源策略限制，在使用跨文档消息通信时，必要的安全意识一定要谨记。

1. 同源策略

由于 Web 浏览器必须遵守同源策略，因此客户端的 Ajax 应用程序一般不能与第三方服务器通信。这一策略规定 JavaScript 代码只能访问其来源服务器上的数据。事实上，这一策略是非常必要的。

同源策略又名同域策略，是浏览器中的主要安全措施。这里的“源”指的是主机名、协议和端口号的组合；我们可以把一个“源”看作是某个 web 页面或浏览器所浏览的信息的创建者。同源策略，简单地说就是要求动态内容（如 JavaScript）只能阅读与之同源的那些 HTTP 应答和 cookies，而不能阅读来自不同源的内容。

2. 跨文档通信安全

一般情况下，我们会通过跨源通信的消息事件中的源来确定消息来源是否可靠。同时也有必要增加事件监听器来监听不可信的干扰消息。一般都会提供一份白名单，以协助浏览器做安全处理，即对比消息来源（origin）是否在可信赖的白名单中。

另外对于传递过来的消息数据，也应该谨慎使用，即便是可靠的消息来源，也应该像对待外部输入时一样慎重。首先要避免传递 HTML 标签数据，因为不恰当的标签数据可能会影响页面的布局。其次是应避免使用 eval()方法来处理传递过来的数据，因为可能会发生不可预期的脚本执行，建议使用 JSON 对象的操作。

如下安全示例应牢记。

```
//不安全：e.data 数据如果包含 HTML 标签，可能会造成页面布局发生变化
element.innerHTML=e.data;
//比较安全
element.textContent=e.data;

//不安全：e.data 数据可能会包含一些不可预期的脚本执行
eval(e.data);
//比较安全
JSON.parse(e.data)
```

14.1.3　使用 postMessage 接口

本节将详细介绍 postMessage 接口的使用方法及相关注意事项。

1．检测浏览器支持情况

在使用 postMessage 接口之前，按照惯例，我们需要检测浏览器是否支持它。检测方法如下：

```
if(typeof window.postMessage === "undefined"){
    alert("您的浏览器不支持 postMessage! ");
}
```

2．发送消息

通过调用目标页面的 window 对象中的 postMessage()函数，可向目标页面发送消息。

```
targetWindow.postMessage("Hello, world", "www.example.com");
```

说明：postMessage 的第一个参数表示发送的数据，第二个参数表示消息传送的目标页面的域（可以使用“*”，表示对于消息传送的目标页面没有域的限制）。

例如，发送消息给当前页面中的 iframe，可以在相应 iframe 的 contentWindow 中调用 postMessage()方法：

```
document.getElementsByTagName("iframe")[0].contentWindow.postMessage("H
ello,world","*");
```

这里 postMessage()方法的第二个参数为“*”，表示不对目标页面的域进行限制。

提示：需要特别注意的是，发送消息的 window 对象是目标页面的 window 对象，而不是当前页面的 window 对象。

3．监听消息事件

监听消息事件是在发送消息的目标页面进行的。通过在 window 对象中添加 message 事件，即可监听发送过来的消息。

【示例 14-1】 监听消息事件并检测有效的来源。

```
//添加白名单
var originWhiteList = ["http://portal.example.com", "http://games.example.
com","http://www.example.com"];
```

```
//检测来源
function checkWhiteList(origin) {
    for (var i=0; i<originWhiteList.length; i++) {
        if (origin === originWhiteList[i]) {
            return true;
        }
    }
    return false;
}
//消息事件处理
function messageHandler(e) {
    if(checkWhiteList(e.origin)) {
        //processMessage(e.data);
    } else {
        //忽略未确认的消息来源
    }
}
//添加消息事件监听器
window.addEventListener("message", messageHandler, true);
```

代码分析：在示例 14-1 中，首先设置了白名单，在接收消息的时候，只接收列入白名单的域发来的消息。checkWhiteList()是针对消息来源进行检测是否列入白名单。最重要的是添加了消息事件监听器 message，并把消息交给 messageHandler()函数来处理。函数 messageHandler()的参数 e 为消息事件 MessageEvent，通过该消息事件，可以访问消息来源（origin）、发送的数据（data）和源窗体对象实例（resorce）等信息。

提示： 在部署跨文档消息传输的时候，监听消息的页面和发送消息的页面应从属于不同的域名，然后再建立两个页面的关联（如通过 iframe 或弹出页面等方式），以方便这两个网页获取到对方的窗口代理，最终实现信息交换。

14.1.4　消息事件接口 MessageEvent

在上一节中，我们介绍了如何使用消息事件来获取消息数据，并用于检测来源的安全。HTML 5 提供了消息事件接口 MessageEvent，用于通信过程中的消息处理。

1. 接口清单

【示例 14-2】 MessageEvent 接口清单。

```
interface MessageEvent : Event {
    readonly attribute any data;
    readonly attribute DOMString origin;
    readonly attribute DOMString lastEventId;
    readonly attribute WindowProxy source;
    readonly attribute MessagePortArray ports;
    void initMessageEvent(in DOMString typeArg, in boolean canBubbleArg, in
    boolean cancelableArg,
        in any dataArg, in DOMString or iginArg, in DOMString lastEventIdArg,
        in WindowProxy sourceArg, in MessagePortArray portsArg);
};
```

2. 接口属性

消息事件接口 MessageEvent 的属性均为只读属性。

- data：数据属性

获取消息中的数据。

- origin：来源属性

获取消息的来源。应用于服务器发送的事件和跨文档消息传输。消息的来源通常是一种格式，包含了主机名和端口号。但不是文档来源的路径或其他不完整的片段标识。

- lastEventId：最后事件的编号

获取最后一个事件的 ID 编号。应用于服务器发送事件，表示最后一个事件的事件源 ID 编号。

- source：源代理属性

获取源窗口的代理。应用于跨文档消息传输。

- ports：端口属性

获取消息的端口数组。常用于跨文档消息传输和消息通道。

3．接口方法initMessageEvent()

消息事件接口 MessageEvent 有一个方法 initMessageEvent()。

initMessageEvent()方法是以一定的方式，初始化为同样名称的 DOM 事件接口方法。

4．MessageEvent接口说明

HTML 5 定义的 MessageEvent 接口也是 WebSockets 和 Workers 的一部分。HTML 5 的通信功能中，用于接收消息的 API 与 MessageEvent 接口是一致的。

在本章的跨文档消息传输的应用中，一般会用到 MessageEvent 接口的 data 属性、origin 属性、source 属性和 ports 属性。其他属性和方法会在后面的章节中涉及，由于它们同属于同一个接口，因此这里一并列出。

14.1.5　实验室：跨文档消息传输示例

本节将创建一个跨文档的消息传输示例，利用 postMessage()方法在不同文档之间传输消息。本节要介绍的案例是把两个跨文档的页面中的数据一起计算，并反馈计算结果。

1．案例简介

该案例包含两个相互传送消息的主体页面和框架页面。其中主体页面 main.html 包含一个表单输入和一个“计算”按钮；框架页面 frame.html 包含一个输入表单和一段文字说明。

主体页面 main.html 的域为“http://yxw740:8081”；框架页面 frame.html 的域为“http://yang:8081”。框架页 frame.html 通过 iframe 标签嵌在主体页面 main.html 中。我们要实现的是让这两个页面通信。

单击主体页面的“计算”按钮，页面会把输入的表单数据传送到框架页面中，并与框架页面的表单数据进行乘法运算。最后，框架页面会把计算结果返回到主体页面中。主体页面接收到回传的计算结果，会把它在相应的位置显示出来。页面结构如图 14-2 所示。

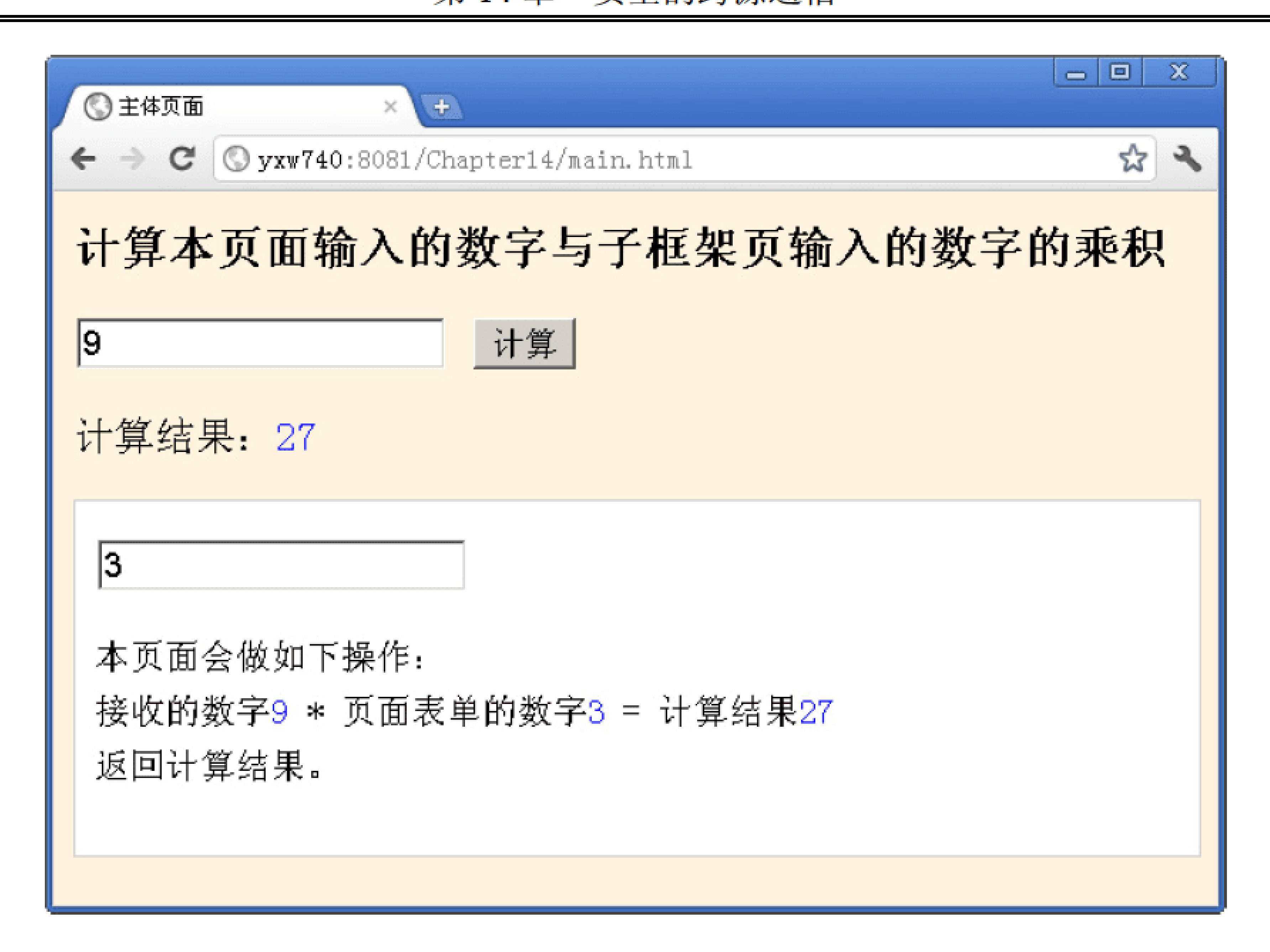


图 14-2　跨文档消息示例

下面就逐步实现所描述的案例。

2．服务器部署

为了提供不同的域名，我们需要部署两个站点，分别为 http://yxw740:8081 和 http://yang:8081。在这里，我们使用更加便捷的方法。

更新 Windows 系统的 hosts 文件（位于 C:\WINDOWS\system32\drivers\etc\hosts）或 Linux 系统的 hosts 文件（位于/etc/hosts），增加两条指向 127.0.0.1 的记录，如下所示：

```
127.0.0.1   yxw740
127.0.0.1   yang
```

这样即可以不同的名称去访问同一个目录下的文件，表现出来的是不同的域，相互之间不能直接调用脚本。

我们选择主体页面使用域“http://yxw740:8081”；框架页面使用域“http://yang:8081”。这一点在后面的代码中会有所体现。

3．主体页面的内容设计

主体页面 main.html 包含一个文本输入框、一个“计算”按钮和一个 iframe 框架，其中该框架页地址指向 http://**yang:8081**/Chapter14/frame.html。

主体页面 main.html 的运行地址为 http://**yxw740:8081**/Chapter14/main.html。

【示例 14-3】 主体页面设计（main.html）。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="uft-8" />
<title>主体页面</title>
<style type="text/css">
```

```
    /* 样式表省略 */
</style>
</head>
<body>
<h1>计算本页面输入的数字与子框架页输入的数字的乘积</h1>
<p>
    <input type="text" id="msg" name="msg" value="" placeholder="请输入数字" />
    <input type="button" id="button" name="button" value=" 计 算 " onclick="sendToFrame()" />
</p>
<p>计算结果：<span id="result"></span></p>
<iframe width="100%" src="http://yang:8081/Chapter14/frame.html" height="140"></iframe>
</body>
</html>
```

4. 主体页面的脚本设计

现在，在主体页面 main.html 中添加脚本实现，需要实现发送消息和接收消息的功能。

- 添加向框架页面发送消息的函数 sendToFrame()，并绑定在“计算”按钮的 onclick 事件上。
- 指定合法的消息来源，即定义了 originWhite。
- 添加消息事件回调函数 messageMainHandler(e)，用于 mssage 事件回调，处理框架页面发来的消息。
- 在 window.onload 事件中，添加浏览器支持检测。如果浏览器支持 postMesage()方法，则添加 message 监听事件。

```
<script type="text/javascript">
/* 发送消息到框架页面 */
function sendToFrame(){
    var win = document.getElementsByTagName("iframe")[0].contentWindow;
/* 获取 iframe 的窗体对象 */
    var val = document.getElementById("msg").value;
    win.postMessage(val,"*");   /* 使用 iframe 窗体对象发送消息给框架页面 */
}
var originWhite = "http://yang:8081";       /* 指定合法的消息来源 */
/* 消息事件回调函数 */
function messageMainHandler(e){
    if(e.origin==originWhite){              /* 验证消息来源 */
        document.getElementById("result").textContent = e.data;
    }else{
        console.log("非法的消息来源！");
    }
}
/* 页面加载处理 */
window.onload=function(){
    /* 检测浏览器支持情况 */
    if(typeof window.postMessage === "undefined"){
        document.getElementById("button").onclick=function(){
            return false;
        }
        console.log("您的浏览器不支持 postMesage！");
```

```
    }else{
        window.addEventListener("message", messageMainHandler, true); /*
添加监听事件 message */
    }
}
</script>
```

5. 框架页面的内容设计

框架页面的内容比较简单，只有一个输入框。输入的内容会与主体页面发来的消息数据进行乘法运算。

【示例 14-3】 框架页面设计（frame.html）。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="uft-8" />
<title>框架页面</title>
<style type="text/css">
    /* 样式表省略 */
</style>
</head>
<body>
<p>
    <input id="name" name="name" type="text" />
</p>
<p>本页面会做如下操作：<br>
接收的数字<span id="getData"></span> * 页面表单的数字<span id="inputData">
</span>
= 计算结果<span id="result"></span><br>
返回计算结果。</p>
</body>
</html>
```

6. 框架页面的脚本设计

在框架页面 frame.html 中添加脚本实现，也需要实现发送消息和接收消息的功能。

- 指定合法的消息来源，即定义了 originWhite。
- 添加消息事件回调函数 messageFrameHandler (e)，用于 mssage 事件回调，处理主体页面发来的消息。
- 添加数据处理函数 sendToMain()。该函数会产生计算结果，并把结果发送给主体页面。
- 在 window.onload 事件中，添加浏览器支持检测。如果浏览器支持 postMesage()方法，则添加 message 监听事件。

```
<script type="text/javascript">
var originWhite = "http://yxw740:8081";          /* 指定合法的消息来源 */
/* 消息事件回调函数 */
function messageFrameHandler(e){
    if(e.origin==originWhite){                   /* 验证消息来源 */
        sendToMain(e.source.window,e.data);      /* 调用数据处理函数 sendTo-
                                                    Main() */
```

```
    }else{
        console.log("非法的消息来源！");
    }
}
/* 数据处理 */
function sendToMain(win,edata){
    var val = parseInt(document.getElementById("name").value);
    var data= parseInt(edata);
    var result = val * data;            /* 计算结果 */
    win.postMessage(result,"*");        /* 将计算结果回发数据给消息的来源窗口
*/
    document.getElementById("getData").textContent = data;
    document.getElementById("inputData").textContent = val;
    document.getElementById("result").textContent = result;
}
/* 页面加载处理 */
window.onload=function(){
    /* 检测浏览器支持情况 */
    if(typeof window.postMessage === "undefined"){
        console.log("您的浏览器不支持 postMesage！");
    }else{
        window.addEventListener("message", messageFrameHandler, true); /*
添加监听事件 message */
    }
}
</script>
```

至此，已经可以实现跨文档消息传输了。打开浏览器，输入 http://yxw740:8081/Chapter14/main.html，在打开的主体页面中，输入数字 9，在框架页面输入数字 3，单击“计算”按钮，运行结果如图 14-2 所示。

14.2　跨源请求——XMLHttpRequestLevel 2

XmlHttpRequest 对象实现了 Ajax 的 Web 应用程序的关键功能，提供了对 HTTP 协议的完全的访问，包括发起 POST 和 HEAD 请求以及普通的 GET 请求的能力，可以同步或异步返回 Web 服务器的响应，并且能以文本或者一个 DOM 文档形式返回内容。

XMLHttpRequestLevel 2 是 XmlHttpRequest 的增强版，添加了一些与时俱进的新功能，如跨站点的请求、进度事件、处理发送和接收字节流等。本节将详细讲解 XMLHttpRequest Level 2 的新增功能。

14.2.1　改进的 XmlHttpRequest 对象

XMLHttpRequest Level 2 仅仅是描述 XmlHttpRequest 对象的一个规范说明。在这个规范当中，描述了从属于该规范的 XmlHttpRequest 对象应该具有的功能。

1. 跨源请求

在过去的 XmlHttpRequest 对象中，仅限于与同源的服务器通信。XMLHttpRequest Level

2 允许通过跨源资源共享（Cross-Origin Resource Sharing，CORS）进行跨站点的 XMLHttpRequest 请求。

如图 14-3 所示的跨源请求服务器示意图，页面 a 属于服务器 A。如果需要请求服务器 B，页面 a 和服务器 B 之间的消息通信就是本节讲的跨源请求。

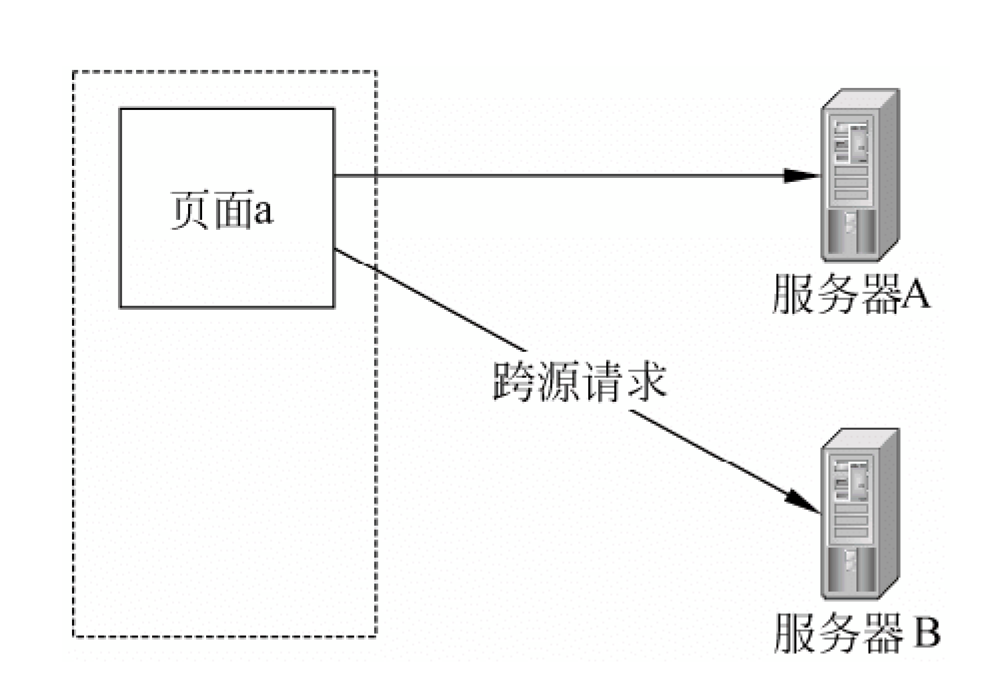


图 14-3　跨源请求服务器示意图

使用跨源的 XMLHttpRequest 请求，使得浏览器中的页面可以访问不同的服务器资源，这为构建非同源服务的 Web 应用程序提供了良好的解决方案。如网站中的天气预报、股票信息、实时汇率等功能，由于自己的服务器无法直接提供如此专业的服务，最好的解决方案就是去专门提供这些服务的服务器请求资源。

另外，随着互联网的服务越来越精细化，把各种服务分别部署在不同的服务器上，不但可以提供专业化的服务，而且便于管理、维护和扩展。用户可以通过浏览器的 XMLHttpRequest 跨源请求，直接访问不同服务器的资源。

2．进度事件

XMLHttpRequest Level 2 为 XMLHttpRequest 对象新增了对进度的一系列响应事件。之前的版本中只有一个相应响应的事件 onreadystatechange，难以为用户提供准确的进度提示服务。

利用一系列新的进度事件，不但可以获取详细的请求进度信息，还可以跟踪资源上载的进度信息，使得 XMLHttpRequest 对象请求资源更加细致化。

3．其他功能

XMLHttpRequest Level 2 的其他新增的 XMLHttpRequest 对象功能还包括处理发送和接收字节流等功能，本书不做详细介绍。

14.2.2　XMLHttpRequestLevel 2 规范说明

为使我们对 XMLHttpRequestLevel 2 规范下的 XMLHttpRequest 对象有更深入的了解，本节将介绍该规范涉及的具体内容。由于会兼容前期的版本，因此很多原有的功能也保留了下来。

1．XMLHttpRequest Level 2接口清单

XMLHttpRequestLevel 2 包含了两个主要的接口，清单如下。

【示例 14-4】 XMLHttpRequestEventTarget 接口清单。

```
interface XMLHttpRequestEventTarget : EventTarget {
    //事件句柄（处理程序）属性
    attribute EventListener onabort;
    attribute EventListener onerror;
    attribute EventListener onload;
```

```
    attribute EventListener onloadstart;
    attribute EventListener onprogress;
};
```

【示例 14-5】 XMLHttpRequest 接口清单。

```
[Constructor] interface XMLHttpRequest : XMLHttpRequestEventTarget {
    //事件句柄（处理程序）属性
    attribute EventListener onreadystatechange;
    //就绪状态
    const unsigned short UNSENT = 0;
    const unsigned short OPENED = 1;
    const unsigned short HEADERS RECEIVED = 2;
    const unsigned short LOADING = 3;
    const unsigned short DONE = 4;
    readonly attribute unsigned short readyState;
    //请求元数据
    attribute boolean withCredentials;
    void open(in DOMString method, in DOMString url);
    void open(in DOMString method, in DOMString url, in boolean async);
    void open(in DOMString method, in DOMString url, in boolean async,
        [Null=Null, Undefined=Null] in DOMString user);
    void open(in DOMString method, in DOMString url, in boolean async,
        [Null=Null, Undefined=Null] in DOMString user, [Null=Null,
        Undefined=Null] in DOMString password);
    void setRequestHeader(in DOMString header, in DOMString value);
    //请求
    readonly attribute XMLHttpRequestUpload upload;
    void send();
    void send(in ByteArray data);
    void send([Null=Null, Undefined=Null] in DOMString data);
    void send(in Document data);
    void abort();
    //响应元数据
    DOMString getAllResponseHeaders();
    DOMString getResponseHeader(in DOMString header);
    readonly attribute unsigned short status;
    readonly attribute DOMString statusText;
    //响应内容体
    void overrideMimeType(mime);
    readonly attribute ByteArray responseBody;
    readonly attribute DOMString responseText;
    readonly attribute Document responseXML;
};
```

在上述的两个接口清单中，XMLHttpRequestEventTarget 接口中仅包含一系列的响应事件，而常用的 XMLHttpRequest 是继承 XMLHttpRequestEventTarget 接口的。

通过接口清单，我们可以对 XMLHttpRequest 对象有一个全新的了解，由于篇幅所限，这里不再对各个属性和方法做具体的说明。

另外 XMLHttpRequestLevel 2 还提供了未完善的接口 XMLHttpRequestUpload，也继承了响应事件的接口 XMLHttpRequestEventTarget，如下所示：

```
interface XMLHttpRequestUpload : XMLHttpRequestEventTarget {
    //供未来使用
};
```

2. 新的响应事件

XMLHttpRequestLevel 2 完善了事件接口 XMLHttpRequestEventTarget。其中包括以下监听事件。

- abort

当请求中断时的事件处理句柄。如调用了 abort()方法。

- error

当请求失败时的事件处理句柄。

- load

当成功完成请求时的事件处理句柄。

- loadstart

当请求开始时的事件处理句柄。

- progress

当正在加载数据或发送数据时的事件处理句柄。

14.2.3 使用新的 XMLHttpRequest 对象

本节将介绍一下如何使用新的 XMLHttpRequest 对象。

1. 检测浏览器支持情况

如果需要使用 XMLHttpRequestLevel 2 下的新的 XMLHttpRequest 对象功能，如跨源请求，则需要检测当前的浏览器是否支持该功能。常用的方法是检测 XMLHttpRequest 对象是否存在 withCredentials 属性。如下所示：

```
var xmlHttp= new XMLHttpRequest();
if(typeof xmlHttp.withCredentials==="undefined"){
    alert("你的浏览器不支持跨源请求");
}else{
    alert("你的浏览器支持跨源请求");
}
```

2. 构建跨源请求

构建跨源请求与构建同源的服务器请求在实现方面基本上类似，只需要指定一个不是同源的请求地址即可。代码如下：

```
xmlHttp.open("GET","http://yang:8081/Chapter14/cors.html",true);
xmlHttp.send(null);
```

关于跨源请求的实现，没有技术上的难点，只有实现上的不同。

3. 添加监听事件

在跨源请求发出之前，通过添加 XMLHttpRequest 对象的监听事件，可以跟踪请求的执行过程。

【示例 14-6】 添加 XMLHttpRequest 对象的监听事件。

```
xmlHttp.onabort=function(e){
    console.log("请求中断");
}
xmlHttp.onerror=function(e){
    console.log("请求失败");
}
xmlHttp.onload=function(e){
    console.log("请求完成，输出请求内容：");
    console.log(xmlHttp.responseText);
}
xmlHttp.onloadstart=function(e){
    console.log("请求开始");
}
xmlHttp.onprogress=function(e){
    console.log("请求中...");
    if(e.lengthComputable){
        console.log(e.loaded+ "/"+e.total);
    }
}
```

与之前常用的 onreadystatechange 事件相比，新增的事件可以更详细地跟踪请求的整个过程。其中通过使用 onprogress 事件，可以完成实时的进度信息，而这一功能在之前是无法做到的。

所有事件的回调函数，都可以获取到 XMLHttpRequestProgressEvent 对象，这是一个进度对象，其中的信息记录了等待发送数据的总量、已发送数据的总量，以及数据总量是否已知的等。

XMLHttpRequest.upload 对象也包含上述事件，以及 XMLHttpRequestProgressEvent 对象。

```
xmlHttp.upload.onprogress=function(e){
    console.log("上载中...");
    if(e.lengthComputable){
        console.log(e.loaded+ "/"+e.total);
    }
}
```

4．服务器部署

要完成跨源的请求，需要依赖两个条件：首先，使用 XMLHttpRequest 对象访问的页面与当前页面是不能同域的；其次，请求的目标服务器能够解析跨源的 XMLHttpRequest 对象请求。所以，我们需要对服务器做一些相关的配置部署。

（1）在 IIS6 中暴露 Access-Control-Allow-Origin，可通过以下步骤：

单击需要启用 CORS 的站点的属性，在“HTTP 头”标签下的自定义 Http 头部分，添加如下信息即可：

```
自定义 HTTP 头名：Access-Control-Allow-Origin
自定义 HTTP 头值：*
```

（2）在 IIS7 中可以复制以下代码到你的站点或应用程序根目录下的 web.config 文件中：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <system.webServer>
   <httpProtocol>
     <customHeaders>
       <add name="Access-Control-Allow-Origin" value="*" />
     </customHeaders>
   </httpProtocol>
 </system.webServer>
</configuration>
```

5. HTTP头

HTTP（HyperTextTransferProtocol）是超文本传输协议的缩写，它用于传送 WWW 方式的数据。HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URI、协议版本，以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。

HTTP 头部消息通常包括客户机向服务器的请求消息和服务器向客户机的响应消息。以下是执行一个跨源请求的 HTTP 头部示例。

【示例 14-7】 跨源的请求头部示例。

```
GET /Chapter14/cors.aspx HTTP/1.1
Host: yang:8081
User-Agent: Mozilla/5.0 (Windows NT 5.2) AppleWebKit/535.2 (KHTML, like
Gecko) Chrome/15.0.874.121 Safari/535.2
Accept: */*
Connection: keep-alive
Cache-Control: max-age=0
Origin: http://yxw740:8081
Referer: http://yxw740:8081/Chapter14/Code14-Test.html
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
Accept-Charset: GBK,utf-8;q=0.7,*;q=0.3
```

【示例 14-8】 跨源的响应头部示例。

```
HTTP/1.1 200 OK
Date: Fri, 23 Dec 2011 01:41:25 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Access-Control-Allow-Origin: http://yxw740:8081
Access-Control-Allow-Credentials: true
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html;
charset=utf-8 Content-Length: 12
```

跨源请求的头部信息会包含 Origin 和 Referer 信息，浏览器会帮助完成。而响应的头部信息中的 Access-Control-Allow-Origin 信息，则是在服务器部署的。

14.2.4　实验室：跨源请求示例

本节将利用 XMLHttpRequest 对象的新增功能来实现跨源的请求。

1．案例简介

本节介绍的案例中的页面，包含一个按钮和用于记录响应信息和进度状态的区域。通过单击按钮事件触发一个跨源的请求，请求的结果将显示在响应信息区域；请求过程中的状态则在进度状态的区域显示。成功执行跨源请求的页面效果如图 14-4 所示。



图 14-4　成功的跨源请求示例

提示：本案例跨源请求的文件仅仅反馈一个字符串“Hello World! ”，所以不再对其进行设计；而响应跨源请求的服务器，认为已经配置完整。

2．界面设计

给页面添加一个按钮作为发起跨源请求的起点，并添加响应信息的记录区域。绑定“请求资源”按钮 onclick 事件为 AjaxRequest()函数。本文件的运行域为“http://yxw740:8081”。

【示例 14-9】　跨源请求示例。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="uft-8" />
<title>跨源请求</title>
<style type="text/css">
    <!-- 省略样式表 -->
</style>
</head>
<body>
<form name="form1" method="post" action="">
    <input name="button" type="button" value="请求资源" onclick=
    "AjaxRequest()" />
    <div class="pad10">响应信息:
        <p id="response"></p>
    </div>
    <div id="progress" class="pad10">进度状态:
        <p>预备</p>
```

```
    </div>
</form>
</body>
</html>
```

3．设计文件脚本

设计 AjaxRequest()脚本函数：新建 XMLHttpRequest 对象，并检测浏览器支持性；添加新增的所有事件处理，以跟踪监视跨源请求的进度信息；发起跨源的请求，请求地址为另一个域“http://yang:8081”下的文件。SetProgress()用于在页面上更新进度信息。

```
<script type="text/javascript">
function AjaxRequest(){
    var xmlHttp = new XMLHttpRequest();
    if(typeof xmlHttp.withCredentials==="undefined"){
        alert("你的浏览器不支持跨源请求");
        return;
    }
    xmlHttp.onabort=function(e){
        SetProgress("请求中断");
    }
    xmlHttp.onerror=function(e){
        SetProgress("请求失败");
    }
    xmlHttp.onload=function(e){
        SetProgress("请求完成.");
        var obj = document.getElementById("response");
        obj.innerText = xmlHttp.responseText;
    }
    xmlHttp.onloadstart=function(e){
        SetProgress("请求开始");
    }
    xmlHttp.onprogress=function(e){
        SetProgress("请求中...");
    }
    xmlHttp.open("get","http://yang:8081/Chapter14/cors.aspx",true);
    xmlHttp.send(null);
}
function SetProgress(str){
    var obj = document.getElementById("progress");
    var p = document.createElement("p");
    p.innerText = str;
    obj.appendChild(p);
}
</script>
```

完成以上步骤，就已经可以实现跨源的请求了。打开浏览器，输入该文件的地址，在显示的界面上单击“请求资源”按钮，运行结果如图 14-4 所示。

14.3　小　　结

本章主要讲解了 HTML 5 跨源通信的概念，重点讲解了发生在不同域的页面之间的跨文档消息传输，以及访问其他服务器资源的跨源请求。本章的难点在于跨源概念的理解。

对于跨文档消息传输中的消息事件接口，本章仅使用了其中的一部分功能，完全理解是比较难的；对于增强的 XMLHttpRequest 对象，在进度控制方面有了全新的进度事件，要实现跨源请求，还需要对相关的服务器做一些必要的配置，这些对于前端人员来也是难点。但是掌握这些对于前端开发人员来说，也意味着将会迎来新的机遇。

下一章将介绍 HTML 5 的 Web Socket 双向通信。

14.4　习　　题

【习题 1】请简要说明一下跨文档消息传输是如何实现的。

【习题 2】消息事件 MessageEvent 中，包含了三个重要的属性 data、origin 和 source。请说明三个属性的含义。

【习题 3】在 HTML 5 中，新的 XMLHttpRequest 对象做了哪些改进？

【习题 4】在 HTML 5 中，新的 XMLHttpRequest 对象有哪些进度事件？

【习题 5】请部署一个支持跨源请求的服务器，并编写一个跨源的异步请求的应用。

第 15 章　强大的 WebSocket 双向通信

在传统的 Web 应用程序中，服务器无法主动地向浏览器推送数据，需要浏览器先发出请求才能返回数据。如果要获取实时的数据，通常会使用轮询等方式不断地请求服务器，但这种做法有着致命的缺陷。所幸的是，HTML 5 规范提供一种全新的双工通信模块——WebSocket，它可以为浏览器与服务器之间提供一个全双工的通信信道，双方可以通过这个信道直接向对方实时地发送数据，而不需要再次请求连接。本章将详细介绍与 WebSocket 相关的概念、基本原理和编程接口。

15.1　WebSocket 概述

下面简单介绍 WebSocket 的概念及面临的问题等相关的内容。

15.1.1　WebSocket 简介

WebSocket 是一种基于一个 TCP 连接全双工的通信技术，主要是在浏览器和服务器之间提供一种双向通信的解决方案。它通过简单的握手协议，建立一个长连接，然后按照协议的规则进行数据的传输。

作为 HTML 5 的新特性的 WebSocket 格外吸引开发人员的注意，它使得浏览器对 Socket 的支持成为可能。Web 开发人员也可以基于 WebSocket 构建一个实时的 Web 应用程序，也使得 Web 应用功能更接近于桌面应用。

WebSocket 包含两个方面的内容：一方面是 WebSocket 协议，主要是用于在浏览器与服务器之间建立通信；另一方面是浏览器中的 WebSocket 编程接口，用于网页的 JavaScript 脚本编程。

WebSocket 协议可以使用任何服务端的编程语言来实现。只有浏览器和服务器都遵循了同样的该协议，才能建立起 TCP 连接，才可以有后续的通信。

WebSocket 编程接口主要用于浏览器客户端的 JavaScript 编程开发。前端开发人员可以通过该接口提供的一些操作，访问提供了 WebSocket 的服务器，从而实现与服务器实时的通信。关于 WebSocket 编程接口，我们会在后面的小节中详细讲解。

15.1.2　实时 Web 应用面临的问题

在 WebSocket 之前，Web 应用全部是基于 HTTP 协议的。浏览器和服务器之间的通信，通常是浏览器向服务器发起请求，然后服务器根据收到的请求信息返回相应的结果。而在

实际的开发中，为了能够及时获取最新的信息，开发者们常常会试图实现实时的 Web 应用。由于在传统的 Web 应用中，服务器不会主动地向浏览器推送信息，所以如何保证客户端和服务器端的信息同步，对开发者来说是一个极大的考验。

目前，所有的实时 Web 应用的实现方式都是一些折中的方案。通常的做法就是使用轮询（Polling），其中比较著名的是 Comet。Comet 技术有两种实现方式：一种是长轮询，另一种是 iframe 流。

轮询是最早的一种实现实时 Web 应用的方案。浏览器以一定的时间间隔向服务器端发出请求，以频繁请求的方式来保持客户端和服务器端的同步。轮询的最大问题是，对于更新数据时间间隔不确定的服务器，会产生大量无用的请求，不但浪费网络带宽，还会白白增加服务器的压力，所以这是一种非常低效的实时方案。

长轮询，是指打开一条连接以后保持，等待服务器推送来数据再关闭的方式。长轮询是对轮询技术的改进和提高，大幅度地减少了无用的请求。然而当服务器的数据更新比较频繁时，与一般的轮询相比，长轮询并没有实质的性能改善。

iframe 流，就是在页面中插入一个隐藏的 iframe，通过其属性 src 向服务器发出一个完整的 HTTP 请求。服务器收到请求后会打开一个保持连接状态的响应，这个连接状态会一直保持下去，永不过期，服务器可以通过这个连接源源不断地向浏览器发送信息。然而，当面对很多这样的长连接时，对服务器资源是一个极大的考验。

综合上述几种方案，都不是真正的实时技术，而且会产生大量的 HTTP 请求和响应，其中的网络开销主要是额外的报头数据。面对大量的 HTTP 请求，服务器的承受能力也是一个极大的考验，随着访问量的增加，服务器的资源开销也会成倍增加。对于开发人员来说，编程实现也非常复杂，需要考虑服务器的承载能力等因素。

15.1.3　WebSocket 的优势

如今，面对实时的 Web 应用，WebSocket 提供了便捷实现方式。与传统的轮询等方式相比，使用新的 WebSocket 有着非常强大且无法超越的优势。

首先，降低了不必要的网络开销。由于每次发送 HTTP 请求都会包含大约 800B 报头信息（HTTP 头）；如果使用 WebSocket 构建应用程序，则每个消息都是一个 WebSocket 帧，仅有 2B 的开销。随着访问用户的增加，WebSocket 网络开销的优势越来越明显。如图 15-1 所示，分别是一千（用例 A）、一万（用例 B）、十万（用例 C）个客户访问量，每个客户轮询一次和使用 WebSocket 帧消息的网络开销对比。

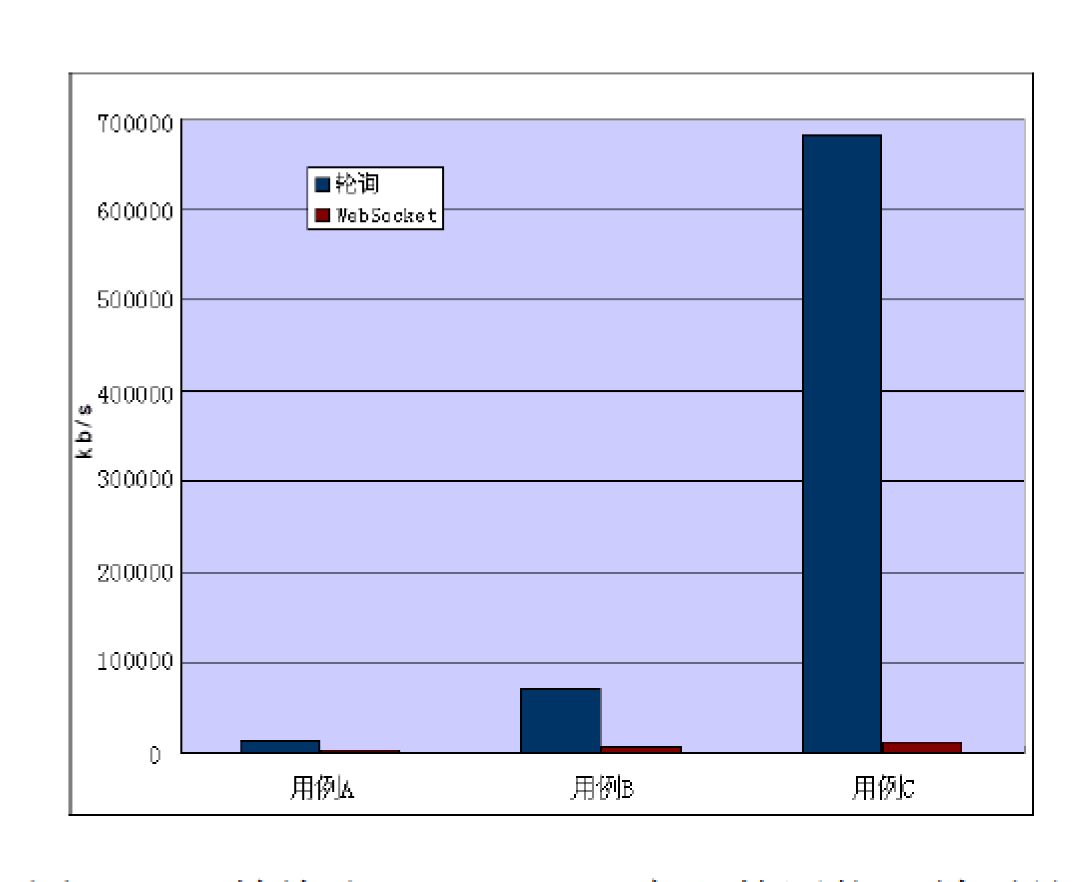


图 15-1　轮询和 WebSocket 实现的网络开销对比

其次，降低了服务器的压力。由于 WebSocket 只有与服务器建立连接的时候发送一次请求，之后的消息传递不需要再次请求，整个过程中，服务器只处理一次请求。与处理轮询方式的大量请求相比，服务器的压力大幅度降低。

再次，信息反馈更加及时。一旦浏览器通过 WebSocket 与服务器建立连接之后，双方就可以直接相互发送消息。与使用 HTTP 请求/响应相比，省去了不必要的延时。

另外，穿越代理和防火墙的能力。由于 WebSocket 是使用 HTTP_CONNECT 代理协议的，浏览器向服务器发送的仍然是一个 HTTP 请求，而这个请求是一个申请协议升级的 HTTP 请求，服务器根据这个与浏览器建立连接，而 HTTP 协议是不受防火墙限制的。所以，基于 WebSocket 的应用，通常具有超强的环境适应能力。

最后，开发简单方便。WebSocket 提供的编程接口避免开发人员与复杂的协议打交道，所以不了解通信协议的开发者，仍然可以开发出基于 WebSocket 的 Web 应用。

15.2 WebSocket 协议

本节介绍一下 WebSocket 的握手协议及各浏览器的支持情况。

15.2.1 WebSocket 握手协议

WebSocket 协议实际上是一个基于 TCP 的协议。WebSocket 协议订立了 HTTP 握手的行为来将已经存在的 HTTP 连接升级为 WebSocket 连接。WebSocket 没有试图在 HTTP 之上模拟 server 推送的通道，而是直接在 TCP 之上定义帧协议。

下面介绍一下 WebSocket 规范。这个规范目前还没有正式形成，尚处于草案阶段，版本更新也比较快。我们选择 draft-ietf-hybi-thewebsocketprotocol-10 版本来描述 WebSocket 协议，因为在最新版本的 Chrome 和 FireFox 浏览器中都支持这一协议。

【示例 15-1】 WebSocket 握手协议。

```
从浏览器到服务器
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 8

从服务器到浏览器
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

这是一个典型的发起请求并得到响应的例子。首先从浏览器向服务器发送的 HTTP 协议头中，包含的“Upgrade: websocket”表示连接协议升级到 WebSocket 协议，其他内容就是 WebSocket 协议所需要的相关握手信息。服务器解析这些协议，并返回一个密钥给浏览器，表明同意建立 WebSocket 连接。

一旦连接建立成功，浏览器和服务器就可以基于双工通信信道相互传递 WebSocket 帧

消息。在网络中，每个帧消息都以 0x00 开头，以 0xFF 结尾，中间采用 UFT-8 编码格式。

关于上述的 WebSocket 协议的完整草案，请参考：http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-10。

15.2.2　浏览器的支持情况

由于 WebSocket 协议还在不断完善中，因此不同浏览器的不同版本所支持的协议都有所差别。如表 15.1 所示为各种浏览器的版本支持情况。

表 15.1　各浏览器版本支持的WebSocket协议

Protocol	Internet Explorer	Mozilla Firefox	Google Chrome	Safari	Opera	NetFront
hixie-75			4	5.0.0		
hixie-76 hybi-00		4.0 (DISABLED)	6	5.0.1	11.00 (DISABLED)	
hybi-06	HTML 5 Labs	dev				
hybi-07		6.0				
hybi-09	HTML 5 Labs					
hybi-10	IE10 developer preview	7	14			

由于 WebSocket 协议不是向下兼容的，因此基于 draft-hixie-thewebsocketprotocol-76 版本的浏览器只能访问基于同样协议的服务器提供的 WebSocket 服务。

15.3　WebSocket 编程接口

本节将介绍一下基于 WebSocket 的脚本编程。

1．编程接口简介

WebSocket 接口提供了一系列的属性、方法和事件。其中接口清单如示例 15-2 所示。

【示例 15-2】 WebSocket 接口清单。

```
[Constructor(DOMString url, optional DOMString protocols),
Constructor(DOMString url, optional DOMString[] protocols)]
interface WebSocket : EventTarget {
    readonly attribute DOMString url;

    //就绪状态
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;
```

```
    //网络
    [TreatNonCallableAsNull] attribute Function? onopen;
    [TreatNonCallableAsNull] attribute Function? onerror;
    [TreatNonCallableAsNull] attribute Function? onclose;
    readonly attribute DOMString extensions;
    readonly attribute DOMString protocol;
    void close([Clamp] optional unsigned short code, optional DOMString
    reason);

    //消息
    [TreatNonCallableAsNull] attribute Function? onmessage;
    attribute DOMString binaryType;
    void send(DOMString data);
    void send(ArrayBuffer data);
    void send(Blob data);
};
```

示例 15-2 所示的接口清单其功能分三个部分：一个是就绪状态，另一个是用于监控网络的，最后一个是发送和接收消息。下面我们基于 WebSocket 的接口清单，分别讲解其中的构造函数、属性、方法和事件。

2. 构造函数

由接口清单可知，该接口的构造函数 WebSocket(url,protocols)带有一个或两个参数。可以使用该构造函数建立一个 WebSocket 的连接对象，方法如下：

```
var socket = new WebSocket(url,protocols);
```

其中，第一个参数 url 指定了要连接的 URL 地址。该连接地址是以“ws://”和“wss://”开始的，分别表示常规的 WebSocket 连接和安全的 WebSocket 连接。

第二个参数 protocols 表示建立连接使用的协议，是可选参数。该参数可以是一个字符串，也可以是一个字符串数组。如果是一个字符串，则相当于是由字符串组成的数组；如果该参数省略，则默认为是空的数组。数组中的每一个字符串均为子协议名称。只有服务器选择了其中的一个子协议，连接才会建立。

3. 接口属性

由接口清单可知，WebSocket 包含了如下 6 个属性。

（1）URL（只读属性）

获取完成构造函数解析的结果地址。

（2）readyState（只读属性）

获取连接的状态，该连接状态有如下 4 个值。

- CONNECTING（数字值 0）：连接尚未构建。
- OPEN（数字值 1）：WebSocket 连接已经建立，并且可以进行通信。
- CLOSING（数字值 2）：正在关闭握手连接。
- CLOSED（数字值 3）：连接已经关闭，或连接没有打开。

其中，当 WebSocket 的连接对象创建完成时，readyState 值被设置为 CONNECTING(0)。

（3）bufferedAmount（只读属性）

获取发送前缓冲的尚未发送的应用程序的字节数。这一部分数据是指发送队列中某一

项应用数据已经开始发送但尚未传送到网络上的部分。

（4）extensions（只读属性）

获取服务器选择的扩展。该属性值初始化的时候是一个空的字符串，但在 WebSocket 连接建立之后，它的值可能会发生变化。

（5）protocol（只读属性）

获取服务器选择的子协议。该属性值初始化的时候是一个空的字符串，但在 WebSocket 连接建立之后，它的值可能会发生变化。

（6）binaryType

获取/设置二进制类型。当 WebSocket 的连接对象创建完成时，binaryType 值被初始化为“blob”。

4．接口方法

由接口清单可知，WebSocket 包含了两个方法，分别用于发送消息和关闭连接。

（1）send()

send()用来发送消息，该方法的用法如下：

```
socket.send(data);
```

该方法中有一个参数 data，表示发送的数据。该数据类型可以是字符串数据、Blob 对象或 ArrayBuffer 对象等。

（2）close()

Close()用来关闭连接，该方法的用法如下：

```
socket.close();
```

在执行关闭操作时，如果 readyState 属性是在 CLOSING（2）或 CLOSED（3）状态，则什么都不做。否则，设置 readyState 属性的值为 CLOSING（2），并触发 close 事件。

5．接口事件

WebSocket 还提供了 4 个可监听的事件，可以实时地跟踪消息的传递、连接的状态和产生的错误。

❑ message

当浏览器收到来自服务器的消息时触发的事件。使用的消息事件接口可参考 14.1.4 节。

❑ open

当 WebSocket 连接已经建立，并且可以进行通信时触发的事件。即 readyState 值为 OPEN（1）时触发。

❑ close

当关闭连接时触发的事件。即 readyState 值为 CLOSING（2）时触发。

❑ error

当有任何的错误产生时触发的事件。

6．从协议反馈的过程

当 WebSocket 连接已经建立后，浏览器需要依次执行下列步骤。

首先，更改 readyState 属性的值为 OPEN（1）；其次，更新 extensions 的值和 protocol 的值（如果不为空时更新）；接着，根据服务器反馈的信息完成握手协议；最后，触发 open 事件。

当 WebSocket 的消息事件收到数据时，浏览器需要依次执行如下步骤。

首先，判断 readyState 属性的值，如果不是 OPEN（1），则中止后续的步骤；其次，使用 MessageEvent 接口，并确保 message 消息事件不起泡、不取消并且没有预设动作；接着，初始化消息事件接口的 origin 属性值为 WebSocket 连接对象的 URL 属性值；之后，根据数据的类型初始化消息事件接口的 data 属性值为相应类型的数据；最后，通过 WebSocket 连接对象调用 message 事件。

15.4　使用 WebSocket 编程

本节介绍 WebSocket 的具体使用方法。

1．检测浏览器的支持情况

与其他 HTML 5 的新特性一样，在使用 WebSocket 之前，需要检测浏览器是否支持该特性。检测方法如下：

```
if (!window.WebSoket) {
    console.log("您的浏览器支持 WebSocket. 您可以尝试连接到服务器!");
} else {
    console.log("您的浏览器不支持 WebSocket。请选择其他的浏览器再尝试连接服务器。");
}
```

2．建立连接

WebSocket 建立连接非常简单。只要提供一个 URL 地址，创建一个新的 WebSocket 实例即可。该 URL 地址是以“ws://”和“wss://”开始的。

```
socket = new WebSocket("ws://localhost:8090/chat”);
```

3．添加状态监听事件和消息监听事件

通过给 WebSocket 对象添加监听事件，可以及时地获取连接状态和接收消息。添加事件后，只需要等待事件的发送，而不用再去轮询。下面的代码为 WebSocket 对象添加了 4 个监听事件。

```
socket.onopen = function(){
    console.log("连接已经建立");
}
socket.onclose = function(){
    console.log("连接已经关闭");
}
socket.onerror = function(){
    console.log("发送错误");
}
```

```
socket.onmessage = function(evt){
    console.log("收到来自服务器的消息："+evt.data);
}
```

4. 发送信息

当 WebSocket 对象处于打开状态时，可以响应 send()方法向服务器端发送数据。发送信息只需要如下操作：

```
socket.send("Hello,World! ”);
```

5. 关闭连接

当需要关闭连接的时候，可以直接调用 WebSocket 对象的 close()方法。在浏览器发送消息或接收消息之后，WebSocket 对象是不会主动关闭连接的，只有调用了 close()方法，浏览器才会主动关闭连接。

```
socket.close();
```

15.5　实验室：构建实时的聊天应用

本节将通过一个聊天示例来巩固前面学习的 WebSocket 知识。

1. 案例简介

本节案例是一个基于 WebSocket 的聊天室。该聊天页面主要包含三个聊天方面的功能：首先，通过单击“连接”按钮，与 WebSocket 服务器建立连接；其次，显示聊天室产生的消息；最后，发送聊天消息。聊天页面如图 15-2 所示。

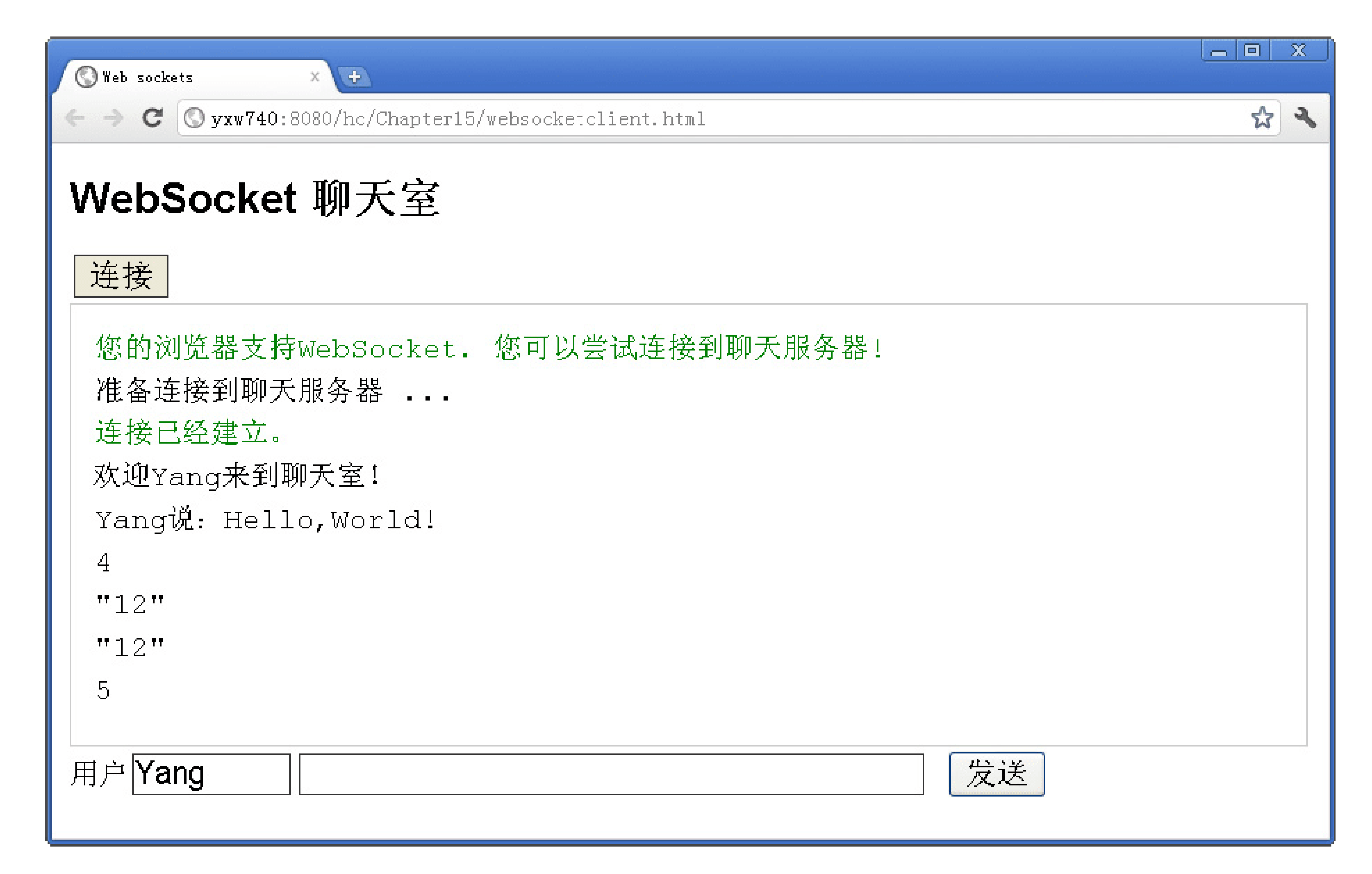


图 15-2　基于 WebSocket 的聊天页面

2．服务器端的部署

一个完整的 WebSocket 应用需要由服务器提供一个 WebSocket 服务。该服务可以使用多种开发语言来开发。服务器端的实现完全需要参考相关的 WebSocket 协议。对于 WebSocket 服务具体实现，这里不做讨论。这里提供几种资源供读者学习使用。

- 本书会提供一份 WebSocket 的服务器端实现的源代码，发布运行后，会提供一个 WebSocket 服务的地址：ws://localhost:8090/chat。
- 这里有 WebSocket 的服务器端的实现，是基于.NET 的：http://superwebsocket.codeplex.com/。运行发布后的文件里的 RunServer.bat，将会提供一个 WebSocket 服务，其默认地址为 ws://localhost:2011/sample。

本节介绍的案例使用本书提供的服务实现，WebSocket 地址为：ws://localhost:8090/chat。

3．设计聊天页面

在聊天页面中，根据案例的描述，设计一个“连接”按钮，用于单击触发与服务器连接；中间设计一个显示区域，用于显示连接过程中产生的消息，以及聊天室其他人发送的消息；最下面设计用于聊天发送信息的姓名、聊天内容和“发送”按钮。

如示例 15-3 所示，“连接”按钮的单击事件绑定了 Connection()处理函数；“发送”按钮的单击事件绑定了 SendData()处理函数。

【示例 15-3】 基于 WebSocket 的聊天页面。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="uft-8" />
<title>Web sockets</title>
<style type="text/css">
    <!-- 省略样式表 -->
</style>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<form id="form1" runat="server">
    <h1>WebSocket 聊天室</h1>
    <div><input type="button" id="ConnectButton" value="连接" onClick=
    "Connection()" /> 连接状态：<span id="state"></span></div>
    <div id='LogContainer' class='container'></div>
    <div>
        用户<input type="text" id="txtName" value="Yang" size="6" /><input
        type="text" id="Data" size="40" />
        <button  id='SendBut'  type="button"  onclick='SendData()'> 发 送
</button>
    </div>
</form>
</body>
</html>
```

4．添加WebSocket代码

首先，在页面加载的时候，判断浏览器是否支持 WebSocket 功能。并设计了 Connection()

处理函数用于与 WebSocket 服务器的连接。

在 Connection()处理函数中，连接完成时，对 WebSocket 对象 ws 绑定 4 个监听事件。其中 open 和 close 用于监听连接状态；error 用于监听错误；message 用于监听接收消息。所有的监听事件都会有消息显示在显示区域。

另外，在 open 事件中，会给服务器发送上线欢迎辞信息，服务器收到消息后，会广播给所有在线人员。

```
<script type="text/javascript">
    var ws;
    var SocketCreated = false;
    window.onload = function(){
        //检测浏览器是否支持 WebSocket
        if (!window.WebSoket) {
            Log("您的浏览器支持 WebSocket. 您可以尝试连接到聊天服务器!", "OK");

        } else {
            Log("您的浏览器不支持 WebSocket。请选择其他的浏览器再尝试连接服务器。", "ERROR");
            document.getElementById("ConnectButton").disabled = true;
        }
    }
    //建立与服务器的连接，并添加监听事件
    function Connection() {
        //如果连接已经存在，则关闭连接。否则建立连接
        if (SocketCreated && (ws.readyState == 0 || ws.readyState == 1)) {
            ws.close();
        } else {
            Log("准备连接到聊天服务器 ...");
            //建立与服务器的连接
            ws = new WebSocket("ws://localhost:8090/chat");
            SocketCreated = true;
            document.getElementById("ConnectButton").value = "断开";
            //添加 open 监听事件，监听打开连接的状态
            ws.onopen = function() {
                Log("连接已经建立。", "OK");
                ws.send("欢迎" + document.getElementById("txtName").
                value+"来到聊天室！");
            }
            //添加 message 监听事件，监听接收消息
            ws.onmessage = function(event) {
                Log(event.data);
            }
            //添加 close 监听事件，监听关闭连接的状态
            ws.onclose = function () {
                Log("连接关闭。", "ERROR");
                document.getElementById("ConnectButton").value = "连接";
            }
            //添加 error 监听事件，监听错误
            ws.onerror = function() {
                Log("WebSocket 错误。", "ERROR");
            }
        }
    }
</script>
```

5. 完善其他代码

添加上一步用到的 SendData()函数和 Log()函数。其中 SendData()函数用于向服务器发送消息；Log()函数用于显示聊天过程中产生的消息。

```
<script type="text/javascript">
    //发送消息给服务器
    function SendData() {
        var obj = document.getElementById("Data");
        if (obj.value != "") {
            ws.send(document.getElementById("txtName").value + "说：" +
            obj.value);
            obj.value = "";
        }
    }
    //显示消息处理函数
    function Log(Text, MessageType) {
        if (MessageType == "OK") Text = "<span style='color: green;'>" + Text
        + "</span>";
        if (MessageType == "ERROR") Text = "<span style='color: red;'>" +
        Text + "</span>";
        var LogContainer = document.getElementById("LogContainer");
        LogContainer.innerHTML += Text + "<br />";
        LogContainer.scrollTop = LogContainer.scrollHeight;
    }
</script>
```

到这里，聊天室的浏览器端就开发完成了。确保 WebSocket 服务器正在运行，然后运行本节介绍的案例的页面，单击“连接”按钮，即可与服务器建立连接。此时，也可以尝试给服务器发送消息。

为了能够演示聊天效果，可以在多个浏览器窗口中打开本页面，或与其他人一起打开本页面。当其中一个浏览器窗口发送消息时，打开该页面的其他窗口或个人，均能收到这个浏览器窗口发送的消息。

15.6　小　　结

本章主要讲解了 HTML 5 的 WebSocket 双向通信。重点讲解了 WebSocket 的概念，以及 WebSocket 的编程接口，并详细讲解了如何使用接口构建一个实时的聊天应用程序。本章的难点在于对 WebSocket 概念的理解，其中涉及的 WebSocket 协议，对很多开发人员来说是一个完全陌生的领域。如果要完全理解和掌握 WebSocket 相关的通信协议，以及相关的服务器方面的部署，则是一个巨大的挑战，需要涉及更多的其他方面的知识。下一章将介绍 HTML 5 的 Web Workers 多线程处理。

15.7　习　　题

【习题 1】请简要说明一下 WebSocket 的优势。

【习题 2】请简要说明一下 WebSocket 握手协议的过程。

【习题 3】下列选项中，哪些是 WebSocket 接口中的事件（多选）：

A．load　　B．close　　C．message

D．processs　　E．error　　F．open

【习题 4】请基于一个 WebSocket 服务器，开发一个实时的聊天页面。

第 16 章　Web 背后——看不见的多线程

一直以来，Web 应用程序都只能在单一的线程中进行。如果需要 Web 应用程序进行长时间的处理如复杂的数学计算等，则会导致 Web 页面长时间处于无法响应的状态，还有可能因为超时而无法处理。HTML 5 新增的 Web Workers 可以让 Web 应用具有多线程处理的能力。通过 Web Workers，可以创建一个专属的处理线程，并且可以再次嵌套子线程；也可以创建一个共享的线程，它允许同域的 Web 应用访问。本章将详细介绍 Web Workers 的各种使用方法及相应的编程接口。

16.1　Web Workers 概述

1. Web Workers简介

为了适应互联网的发展，HTML 5 提出了 Web Workers 的概念，以实现 Web 应用的多线程处理。基于 Web Workers，可以在后台创建一个运行脚本的线程，该线程的运行独立于任何页面。

使用 Web Workers 创建的独立的线程，可以长时间地去执行一个任务。在线程执行任务期间，不会因为用户的单击操作或其他用户界面的交互行为而中断。同样，也不会因为线程还在执行中，造成用户界面无法响应操作。

一般情况下，线程都是做一些重量级的处理，也是无法及时得到反馈的处理，这些交给线程处理，是最合适不过的。因此，这样的处理常常是长时间处于执行的活跃状态，并伴随着大量的 CPU 性能消耗和大量的内存消耗。

关于线程的概念，在其他很多编程语言中都有涉及。随着越来越多的应用处理都交给 Web 页面来处理，HTML 5 增加了线程处理功能，提高了 JavaScript 的编程能力。然而与很多编程语言的线程处理不同的是，使用 JavaScript 编写多线程比较简单，因为很多浏览器已经帮我们做了很多工作。

2. Web Workers线程特征

首先，Worker 线程不能操作 window 对象和 document 对象。如果在线程文件中使用了 window 对象或 document 对象，则会发生错误。

其次，Worker 线程在本质上属于系统线程，是线程执行的一种方式。

最后，HTML 5 还为 Worker 线程提供了范围接口，规范了 Worker 线程的操作范围。例如，在 Worker 线程中可以使用 setTimeout()，clearTimeout()，setInterval()，clearInterval() 等函数。

3. Web Workers体系结构

在最新的 Web Workers 规范中，包含了两种线程：一种是专属线程 Dedicated Worker，另一种是共享线程 Shared Worker。这两种线程有着不同的用途，并且使用不同的接口来实现。

专属线程通常是在一个页面中创建的，并且该线程只能同创建它的页面进行通信。

而共享线程通常是由其中的一个页面创建的，其他页面也可以连接到该线程，使用该线程中的资源和信息。

Web Workers 规范中，不仅包含了专属线程和共享线程的接口，而且包含与线程处理相关的错误处理接口，以及线程使用的导航、本地属性和工作单元等接口。

下面将基于该体系结构逐步讲解。

16.2 专属线程

专属线程（Dedicated Worker），通常是由创建它的页面负责与之通信的，也是较早形成规范的线程处理方式。专属线程使用的是 Worker 对象。

16.2.1 专属线程的基本用法

本节将详细介绍专属线程的基本用法。为了便于说明，我们创建了一个页面 Code16-1.html 和一个线程脚本文件 Code16-1.js。专属线程的用法非常简单，只需要在页面中创建一个 Worker 对象，并指定线程脚本文件即可。在线程通信方面，仍然使用 postMessage ()方法和 message 事件。

1. 检测浏览器的支持情况

与 HTML 5 的其他新特性一样，在使用 Worker 对象之前，需要检测浏览器是否支持该特性。检测方法如下：

```
if(typeof Worker === "undefined"){
    alert("浏览器不支持 Web Workers");
}
```

由于专属线程使用的是 Worker 对象，因此我们只需使用 typeof 运算符来返回 window 对象的 Worker 属性即可。如果浏览器不支持专属线程处理，则返回的是"undefined"。

2. 创建专属线程

在使用 Worker 对象创建专属线程时，需要为 Worker 对象的构造函数提供一个 JavaScript 脚本文件的 URL 地址，该脚本文件中包含了线程中所要执行的代码。需要注意的是，脚本文件的 URL 地址可以是相对地址或者绝对地址，但该 URL 地址受浏览器的同源策略限制。

```
var worker = new Worker("Code16-1.js");
```

实例化 Worker 对象，即创建了一个专属线程。"Code16-1.js"是线程执行的脚本文件。

3．给线程添加监听消息事件

如果线程中有消息反馈，可以通过添加 Worker 对象的 message 事件来监听从线程发来的消息，如下所示：

```
worker.addEventListener('message', function(e) {
    //处理线程 worker 发来的消息
}, true);
```

也可以使用 Worker 对象的 onmessage 事件句柄的方法，如下所示：

```
worker.onmessage = function(e) {
    //处理线程 worker 发来的消息
}
```

4．向线程中发送消息

使用 Worker 对象的 postMessage ()方法，可以向线程中发送消息，如下所示：

```
worker.postMessage("Yang");
```

这里假设向线程中发送一个姓名为"Yang"的字符串。

5．编写线程处理的脚本文件

在后台的线程中是不能访问页面文档或窗口对象的。如果使用了 window 对象或 document 对象，则会发生错误。这里，我们假设线程的脚本做这样的处理：当收到页面发来的消息后，处理消息并把反馈的结果发回页面。

由于线程的脚本文件中涉及消息的接收与发送，因此仍然使用了 postMessage ()方法和 message 事件，如下所示。

```
onmessage = function(e){              //监听页面发来的消息
    var val = "Hello,";
    postMessage(val+e.data);          //向页面发送消息
}
```

通过在文件中直接添加 message 事件，即可监听页面发来的消息；直接使用 postMessage ()方法即可向页面发送消息。

6．在线程中加载多个文件

对于由多个 Jsvascript 脚本文件组成的 Web 应用程序来说，可通过包含<script>元素的方式来同步加载相关的脚本文件。由于在后台的线程中不能访问 document 对象，因此不能采用包含<script>元素的方式。

Web Workers 提供了另外一种导入其他文件的方法——importScript。在脚本文件中导入其他脚本文件，如下所示：

```
importScript("xxx.js");
```

导入的脚本文件只会在已有的 Worker 对象中加载和执行。也可以同时导入多个脚本文

件，如下所示：

```
importScript("xxx.js","yyy,js");
```

7. 监听线程错误

如果执行的线程出现错误，是无法直接反馈到页面上的。不过，我们可以给 Worker 对象添加错误监听事件，来监听线程中出现的错误。特别是调试的时候，使用这种方法监听事件非常有效。和添加 message 事件一样，我们也可以给 Worker 对象添加 error 事件以监听线程中的错误。如下所示。

```
worker.addEventListener('error', function(e) {
    console.warn(e.message,e);
}, true);
```

也可以使用 Worker 对象的 onerror 事件句柄的方法，如下所示。

```
worker.onerror = function(e) {
    console.warn(e.message,e);
}
```

8. 一个简单的线程示例

页面中有一个输入姓名的文本框和一个按钮。当单击按钮时，会调用 Greeting()函数，该函数会新建一个线程，并把输入的姓名发送给线程；线程接收到姓名，会发回一个欢迎辞；页面监听到线程的消息，会把消息显示在页面上。

【示例 16-1】 示例的页面文件。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Web Workers</title>
<script type="text/javascript">
function Greeting(){
    //检测浏览器支持性
    if(typeof Worker === "undefined"){
        console.log("浏览器不支持 Web Workers");
        return;
    }
    //创建专属线程
    var worker = new Worker("Code16-1.js");
    //监听线程的消息
    worker.onmessage = function(e){
        document.getElementById("msg").innerText= e.data;
    }
    //监听线程的错误
    worker.onerror = function(e){
        console.warn(e.message,e);
    }
    var val = document.getElementById("Name").value;
    //向线程发送消息
    worker.postMessage(val);
}
</script>
```

```
</head>
<body>
<input type="text" id="Name" value="" />
<input type="button" value="欢迎辞" onClick="Greeting()" />
<span id="msg"></span>
</body>
</html>
```

示例的线程脚本文件如下所示：

```
onmessage = function(e){          //监听页面发来的消息
    var val = "Hello,";
    postMessage(val+e.data);      //向页面发送消息
}
```

运行结果如图 16-1 所示。



图 16-1　一个简单的线程示例

本节的示例仅作学习之用。实际上，专属线程可以长时间地处理一个后台任务。

16.2.2　多个线程嵌套

线程中可以嵌套子线程，这样我们就可以把较大的线程处理分解成多个子线程，在每个子线程中，各自完成相对独立的部分。

线程嵌套的方法，就是在一个线程中创建另外一个子线程。在线程文件中创建子线程的方法如下：

```
var subworker = new Worker('subworker.js');
```

这样，我们可以像操作在页面中创建的主线程一样，去操作这个线程，如添加消息监听事件和错误监听事件等。

提示：有些支持 Web Worders 线程的浏览器，不一定也支持线程嵌套，如 Chrome 目前的版本就不支持线程嵌套。所以在使用线程嵌套时，有必要先检查一下浏览器是否支持。

多个线程的嵌套有两种方式：单层线程嵌套和多层线程嵌套。

1．单层线程嵌套

单层线程嵌套，主要是在一个主线程中，创建另外一个子线程，以完成独立的工作。由于是多个线程同时处理，因此执行起来是非常高效的。

假设我们在主线程中提供一组数字，分别求出小于它们的最大素数。那么这个求素数的功能，就可以交给子线程来执行。我们需要编写三个文件：Web 网页、主线程文件、子

线程文件。

首先，我们编写 Web 网页（Code16-2page.html），用于创建一个 Worker 的主线程（即专属线程），并监听从主线程中返回的消息，最终显示在页面上。

【示例 16-2】 单层线程嵌套示例。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>线程嵌套示例</title>
</head>
<body>
<p>结果:
    <output id="result"></output>
</p>
<script type="text/javascript">
    var worker = new Worker('Code16-2worker.js');
    worker.onmessage = function (evt){
        document.getElementById('result').innerHTML += "<br />"+
        JSON.stringify(evt.data);
    }
    worker.onerror = function(e){
        console.log(e.message);
        worker.terminate();
    }
</script>
</body>
</html>
```

接着，编写主线程文件（Code16-2worker.js）。在主线程中，我们通过循环 10 次的方式提供了 10 个数字，并分别创建一个子线程，用于计算小于该数字的最大素数。当收到子线程的消息时，立即发送给应用页面，代码如下：

```
//基本设置
var num workers = 10;
var step = 1000;
//开始添加子线程
if(typeof Worker !== "undefined"){
    //开始添加子线程
    for (var i = 0; i <= num workers; i += 1){
        var subworker = new Worker('Code16-2subworker.js');
        subworker.onmessage = storeResult;
        subworker.postMessage(i * step);
    }
}else{
    postMessage("浏览器不支持线程嵌套！");
}
//结果处理
function storeResult(e){
    postMessage(e.data);    //向页面发送消息！
}
```

最后，编写子线程文件（Code16-2subworker.js）。该子线程接收主线程发来的数字，并求出小于该数字的最大素数。代码如下：

```
//求范围内的最大素数
```

```
onmessage = function (e) {
    var num = 1 * e.data + 1;
    search : while (num>1) {
        num --;
        isprime = false;
        for (var i = 2,x=Math.sqrt(num) ; i <= x; i++){
            if (num % i == 0){
                continue search;
            }
        }
        postMessage(e.data + "以内的最大素数是：" + num);
        break;
    }
}
```

运行页面结果如图 16-2 所示。

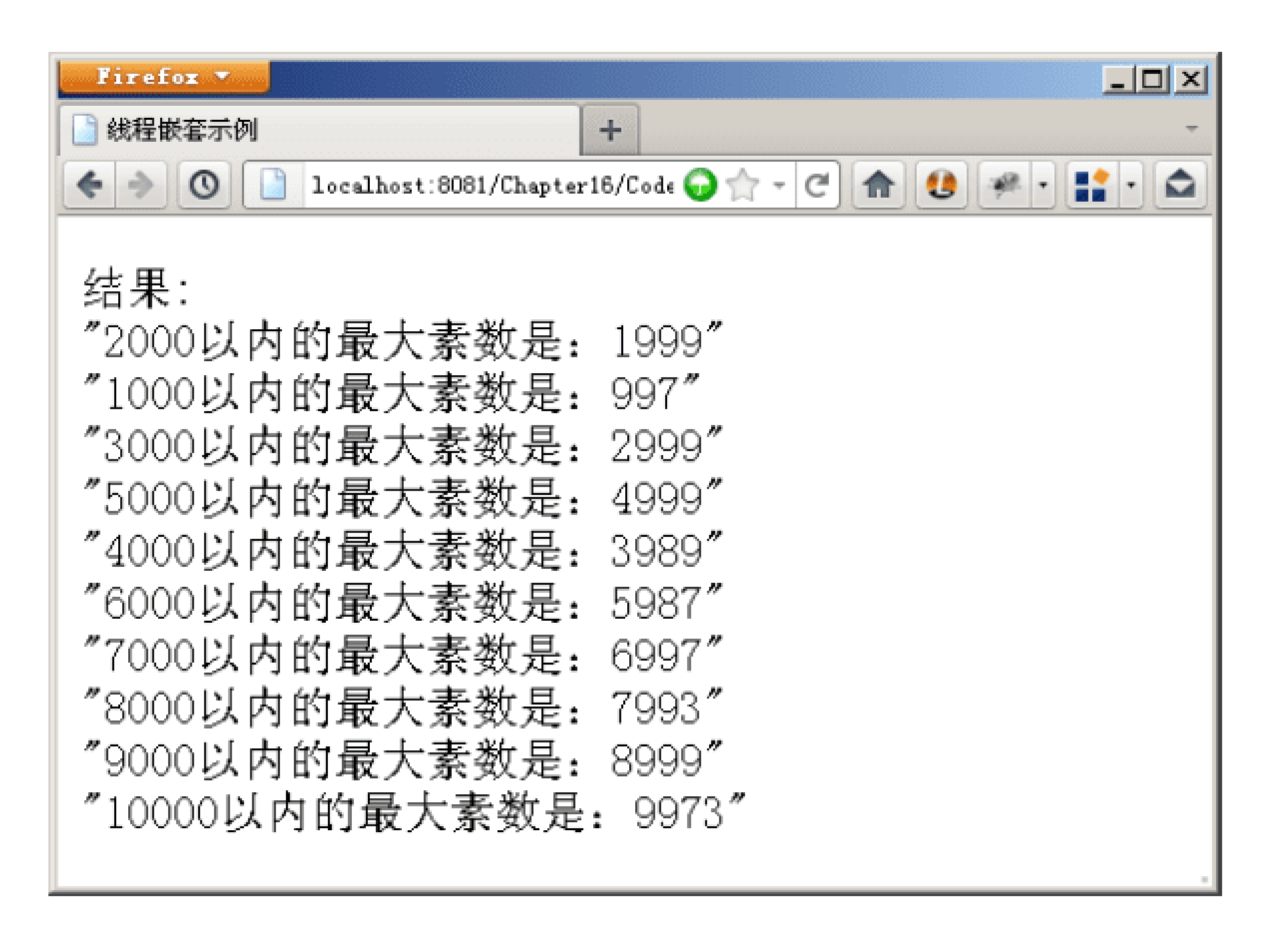


图 16-2　单层线程嵌套示例

代码分析：在示例 16-2 中，主线程连续创建了 10 个子线程，并分别添加了监听消息事件。在如图 16-2 所示的执行结果中，消息返回的顺序与线程创建的顺序是不一致的，说明创建的这些线程是并行执行的。

2. 多层线程嵌套

多层线程嵌套，主要是指主线程中会创建多个子进程，并且在子进程间进行数据交互。与单层线程嵌套相比，多层线程嵌套在代码编写方面是层层嵌套的。

要在多层线程嵌套中实现子线程之间的数据交互，基本上遵循如下步骤。

（1）在主线程中创建子线程，可选择向该子线程发送数据。

（2）执行子线程中的任务，并把执行结果发回主线程。

（3）主线程监听到上一个子线程发来的消息数据后，即创建下一个子线程，并把上一个子线程发来的消息传递给下一个子线程，然后再执行下一个子线程，如此循环嵌套。

（4）主线程接收最后一个子线程发来的消息。

这样，就实现了多层线程嵌套。下面我们用一个样例来了解多层线程嵌套的实现。

【示例 16-3】 多层线程嵌套样例。

```
//创建第一个子线程
var subworker = new Worker('subworker.js');
```

```
subworker.postMessage(100);
//监听第一个子线程的消息
subworker.onmessage = function(e){
    //创建第二个子线程
    var subworker2 = new Worker('subworker2.js');
    //把第一个线程发来的数据发送到第二个线程
    subworker2.postMessage(e.data);
    //监听第二个子线程的消息
    subworker2.onmessage = function(e){
        postMessage(e.data);  //向页面发送消息
    }
}
```

值得注意的是，多层线程嵌套并不是在子线程中再创建子线程，而是所有的子线程都是在主线程中以嵌套的方式创建的，并实现一个子线程向另一个子线程传递数据。

16.2.3　实验室：专属线程中的异步请求

本小节将演示一下专属线程的复杂应用，在一个子线程中发起异步数据请求，并将数据传递到另一个子线程中去处理。这里用到了多层线程嵌套的实现方法。

1. 案例简介

在本节介绍的案例中，Web 页面会创建一个主线程，主线程会再创建一个用于向服务器发起异步请求的子线程，并在该子线程的消息监听事件中创建另一个子线程，用于处理异步请求获得的数据。

整个案例将会用到 5 个文件，分别如下。

- 数据文件（Code16-4xhr.txt）：异步请求的服务器文件，为简单起见，该文件中仅保存了一个数字数组。
- 页面文件（Code16-4page.html）：Web 应用的入口页面，该页面会创建一个主线程。
- 主线程文件（Code16-4worker.js）：该文件会创建两个子线程，并负责把一个子线程的数据传递给另外一个子线程。
- 异步请求子线程文件（Code16-4xhrWorker.js）：主要负责向服务器端发起异步请求，并返回请求的数据。
- 数据处理子线程文件（Code16-4rangeWorker.js）：接收数组数据并处理，以获取数组中的最大值和最小值。

在整个线程的数据交互过程中，我们使用的是结构化的 JSON 数据，以实现更便捷的操作。假设数据文件中是一个数字数组的字符串“[2,1,7,4,9,8,5,3,6] ”，则运行的结果将如图 16-3 所示。

图 16-3　线程中的异步请求

2. 页面文件设计

在页面文件（Code16-4page.html）中负责创建主线程，监听主线程的消息，并把消息

内容显示出来。

【示例 16-4】 线程中的异步请求。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>使用线程处理异步请求的数据</title>
<style type="text/css">
output{color:#006699;border:1px solid #CCCCCC;padding:5px;display:block;}
</style>
</head>
<body>
<p>线程处理结果:
    <output id="result"></output>
</p>
<script type="text/javascript">
    //创建主线程
    var worker = new Worker('Code16-4worker.js');
    //添加消息监听事件
    worker.onmessage = function (evt){
        var json = JSON.parse(evt.data);            //JSON 数据处理
        var obj = document.getElementById('result');
        obj.innerHTML += "最大值: " + json.maxValue;
        obj.innerHTML += "; 最小值: " + json.minValue;
    }
    //添加错误监听事件
    worker.onerror = function(e){
        console.log(e.message);
        worker.terminate();
    }
</script>
</body>
</html>
```

3. 主线程文件设计

在主线程文件（Code16-4worker.js）中，使用的是多层线程嵌套的方案。首先创建一个 xhrWorker 子线程（负责异步请求服务器数据）；其次在 xhrWorker 线程的消息监听事件中创建另外一个 rangeWorker 子线程（接收数组数据并处理）；最后在 rangeWorker 子线程的消息监听事件中向页面发送接收到的消息。

```
//创建异步请求子线程
var xhrWorker = new Worker('Code16-4xhrWorker.js');
//监听异步请求子线程的消息
xhrWorker.onmessage = function(e){
    //创建数据处理子线程
    var rangeWorker = new Worker('Code16-4rangeWorker.js');
    //传递数据
    rangeWorker.postMessage(e.data);
    //监听数据处理子线程的消息
    rangeWorker.onmessage = function(e){
```

```
        postMessage(e.data);  //向页面发送消息
    }
}
```

4．异步请求子线程文件设计

在异步请求子线程文件（Code16-4xhrWorker.js）中，向服务器发起异步请求，并将请求到的文件发送给主线程。

```
//异步请求数据
var xmlHttp = new XMLHttpRequest();
xmlHttp.onerror=function(e){
    postMessage(null);
}
xmlHttp.onload=function(e){
    postMessage(xmlHttp.responseText);
    close();
}
xmlHttp.open("get","Code16-4xhr.txt?"+(new Date()).getTime(),true);
xmlHttp.send(null);
```

5．数据处理子线程文件设计

在数据处理子线程文件（Code16-4rangeWorker.js）中，接收主线程发来的消息，并作为数组进行处理。使用循环的方式求出数组中的最大值和最小值，并将获得的结果放在一个对象中返回。

```
//求数组的最大值和最小值
onmessage = function (e){
    var arr = JSON.parse(e.data);
    var result={
        maxValue:0,
        minValue:0
    };
    if(arr.length>0){
        result.maxValue = arr[0];
        result.minValue = arr[0];
    }
    for(var i = 0, n = arr.length; i < n; i++){
        //求最大值
        if(result.maxValue < arr[i]){
            result.maxValue = arr[i];
        }
        //求最小值
        if(result.minValue > arr[i]){
            result.minValue = arr[i];
        }
    }
    postMessage(JSON.stringify(result));
    close();
}
```

至此，就完成了整个案例，运行结果如图 16-3 所示。这个案例也说明了在 Worker 线程中可以发起异步请求。

16.3　共 享 线 程

共享线程（Shared Worker）可以同时有多个页面的线程连接。共享线程与专属线程的使用对象不同，它是由 SharedWorker 对象创建的。

16.3.1　共享线程的基本用法

本节将详细介绍共享线程的基本用法。要创建共享线程，需要在页面中创建一个 SharedWorker 对象，并指定线程脚本文件。在线程通信方面，仍然使用 postMessage ()方法和 message 事件。

1．检测浏览器的支持情况

与其他 HTML 5 的新特性一样，在使用 SharedWorker 对象之前，需要检测浏览器是否支持该特性。检测方法如下：

```
if(typeof SharedWorker === "undefined"){
    alert("浏览器不支持共享线程！");
}
```

由于共享线程使用的是 SharedWorker 对象，因此只需使用 typeof 运算符来返回 window 对象的 SharedWorker 属性即可。如果浏览器不支持共享线程处理，则返回的是"undefined"。

2．创建共享线程

使用 SharedWorker 对象创建共享线程与使用 Worker 对象创建专属线程的方法基本一致，也需要提供一个 JavaScript 脚本文件的 URL 地址，该脚本文件中包含了线程中所要执行的代码。脚本文件的 URL 地址也可以是相对地址或者绝对地址，同样受浏览器的同源策略限制。

```
var worker = new SharedWorker ("Code16-5sharedWorker.js");
```

实例化 SharedWorker 对象，即创建了一个可以共享的线程。" Code16-5sharedWorker.js"是共享线程执行的脚本文件。

3．给线程添加监听消息事件

与专属线程一样，共享线程也使用 message 事件监听线程消息。但不同的是，共享线程是使用 SharedWorker 对象的 port 属性来与线程通信的，如下所示：

```
worker.port.onmessage = function(e) {   //注意：这里使用 port 属性！
    //处理线程 worker 发来的消息
}
```

也可以使用添加事件的方式，如下所示：

```
worker.port.addEventListener('message', function(e) {
```

```
    //处理线程 worker 发来的消息
}, false);
worker.port.start();   //注意：当使用 addEventListener 时，需要启动端口
```

这里有两个需要注意的地方：一个是事件是附加在对象的 port 属性上的；另一个是当使用 addEventListener 函数来添加事件时，需要使用 worker.port.start()来启动端口。

4．向线程中发送消息

使用 SharedWorker 对象的 port 属性向共享线程发送消息，如下所示：

```
worker.port.postMessage("Yang");
```

与共享线程的通信都是由 SharedWorker 对象的 port 属性来完成的。

5．编写线程处理的脚本文件

在共享线程中，首先使用 connect 事件监听来自不同用户的连接，然后才能在 connect 事件处理程序中为各个连接建立各自的消息监听和消息发送的通信机制，如下所示：

```
var count = 0;  //全局变量
onconnect = function(e) {                          //监听新链接
count += 1;
var port = e.ports[0];
    port.postMessage("欢迎你！ 这是新的链接#" + count);
    port.onmessage = function(e) {                 //为该连接添加消息监听事件
        port.postMessage("你好，" + e.data);       //向该连接的页面发送消息
    }
}
```

在线程文件中，每个连接与线程之间的通信是在 connect 事件中进行的，所以各个通信之间是相互独立的。但是它们可以共享全局的信息（如变量 count）。

另外，共享线程也可以通过 importScript 来导入其他脚本文件。

16.3.2　实验室：共享线程使用示例

与专属线程相比，共享线程避免了线程的重复创建和销毁的过程，降低了系统性能的消耗，如 CPU 处理其调度、内存资源的占用及回收等。在一些编程语言中会有线程池的概念，而线程池也是共享线程的一种应用。

我们使用前面介绍的步骤，实现一个共享线程的示例。该线程会提供一个连接计数器以记录连接数，每当有一个新的连接时，计数器就增加 1，并反馈到当前连接页面。

【示例 16-5】 示例的页面文件。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>共享线程</title>
</head>
<body>
<input type="text" value="" onchange="worker.port.postMessage(this.
value);return false;" />
```

```
<pre id="log">日志:</pre>
</body>
</html>
<script type="text/javascript">
    if(typeof SharedWorker === "undefined"){
        alert("浏览器不支持共享线程");
    }
    var log = document.getElementById('log');
    //创建 SharedWorker 对象
    var worker = new SharedWorker("Code16-5sharedWorker.js");
    worker.port.addEventListener('message', function(e) {
                                                    //注意：这里使用 port 属性
        log.textContent += '\n' + e.data;
    }, false);
    worker.port.start(); //注意：当使用 addEventListener 时，需要启动端口
    worker.port.postMessage("初始化"); //发送消息
</script>
```

共享线程的脚本文件文件如下所示：

```
var count = 0; //计数器
//监听新链接
onconnect = function(e) {
    count += 1;
    var port = e.ports[0];
    //向该连接的页面发送消息
    port.postMessage("欢迎你！ 这是链接" + count);
    //为该连接添加消息监听事件
    port.onmessage = function(e) {
        port.postMessage("线程收到你的消息： " + e.data);
    }
}
```

运行结果如图 16-4 所示。

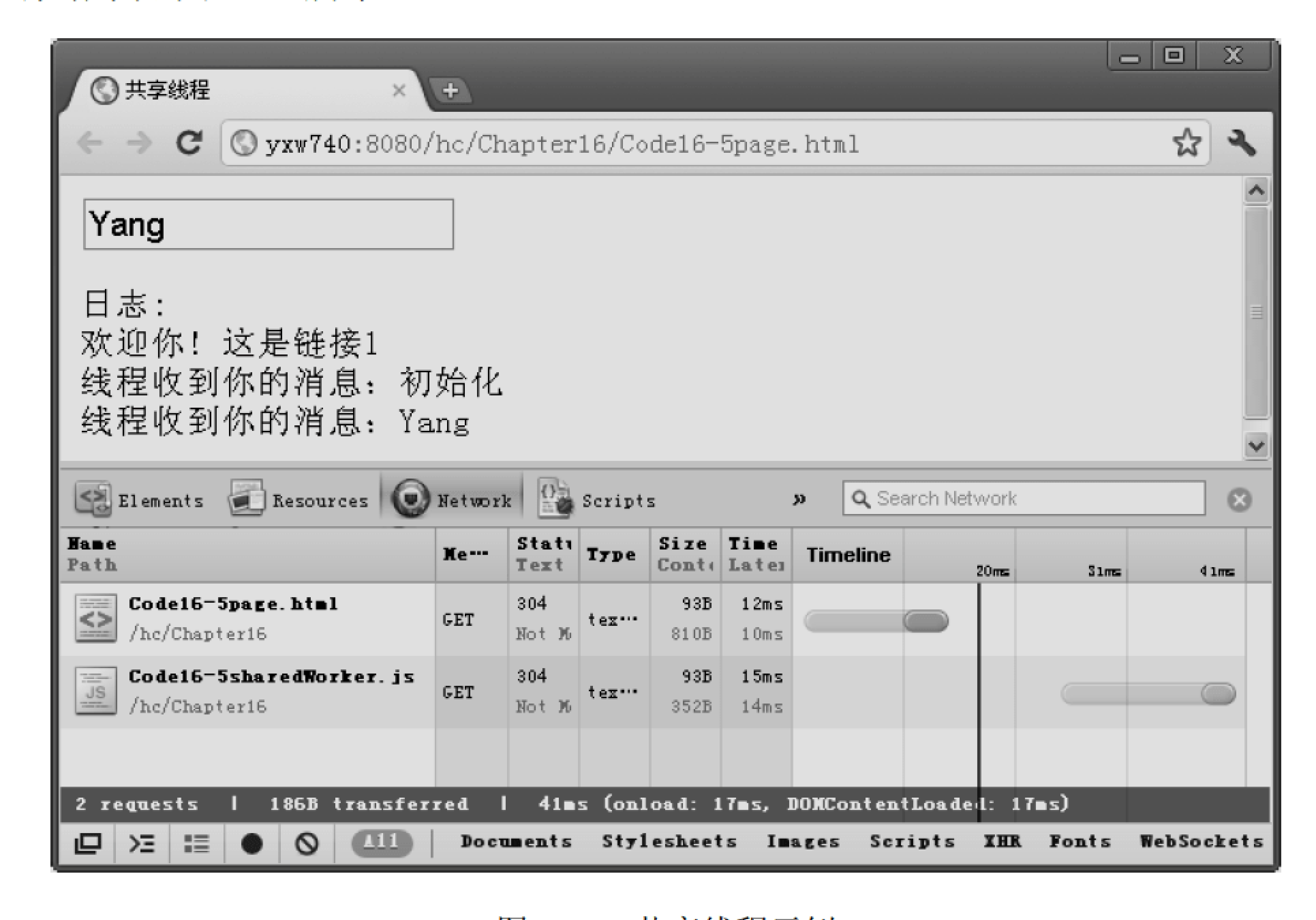


图 16-4　共享线程示例

再次打开该页面，与如图 16-4 所示的页面同时运行，结果如图 16-5 所示。

图 16-5　同时运行的第二个页面

示例说明：由两次运行的结果可见，再次打开的页面如图 16-5 所示，返回的计数器显示为 2，说明是连接到共享进程的第二个连接；由于第一次运行页面创建了共享进程，因此第二次运行页面就没有再次访问线程脚本文件，而是直接连接到已经存在的共享进程；打开的两个页面可以分别与共享进程进行通信，这些通信不会相互干扰；由于计数器是全局变量，因此可以在各个连接到该进程的页面共享该信息。

16.4　Web Workers 接口框架解析

HTML 5 提供了 Web 线程的一系列接口，这些接口大致上可以划分为线程外部的接口和线程内部的接口。

16.4.1　线程外部的接口

线程外部的接口，即操作线程的接口，主要描述了 Web 线程有哪些操作。这里包括专属线程接口和共享线程接口，它们是一脉同宗的，都实现了一个共同的抽象线程接口。

1．抽象线程（AbstractWorker）接口清单

抽象线程接口，是抽象了专属线程和共享线程的共同特征而形成的接口，定义了专属线程和共享线程的必备功能。接口清单如下：

```
[NoInterfaceObject]
interface AbstractWorker {
    [TreatNonCallableAsNull] attribute Function? onerror;
};
```

抽象线程接口定义了一个错误事件 error，表明这个事件必须被支持。

2．专属线程（Dedicated Worker）接口清单

专属线程接口在实现抽象线程的基础上定义了特有的功能。比如接收信息、发送信息及终止线程等操作。接口清单如下：

```
[Constructor(DOMString scriptURL)]
interface Worker : EventTarget {
    void terminate();
    void postMessage(any message, optional sequence<Transferable>
    transfer);
    [TreatNonCallableAsNull] attribute Function? onmessage;
};
Worker implements AbstractWorker;
```

专属线程的 Worker 对象是一个隐含了 MessagePort 的对象，即实现了 postMessage()方法和 message 事件等。由于专属线程实现了抽象线程接口，因此默认有 error 事件。具体说明如下。

- ❑ terminate()方法：调用该方法，可以终止一个运行中的线程。
- ❑ postMessage()方法：用于向线程发送消息，消息可以是字符串，也可以是结构化的数据（如 Json）。
- ❑ message 事件：用于监听线程发来的消息。

3．共享线程（Shared Worker）接口清单

共享线程接口在实现抽象线程的基础上，也定义了特有的功能。比如提供了 port 特性，通过该特性可以实现消息的收发等功能。接口清单如下：

```
[Constructor(DOMString scriptURL, optional DOMString name)]
interface SharedWorker : EventTarget {
    readonly attribute MessagePort port;
};
SharedWorker implements AbstractWorker;
```

共享线程的 SharedWorker 对象只有一个 port 属性。port 属性属于 MessagePort 类型，所以在共享线程中的通信都是基于 port 属性来实现的。另外，SharedWorker 对象实现了抽象线程接口，所以默认包含 error 事件。

4．MessagePort接口清单

MessagePort 接口定义了消息的收发功能，以及消息的收发功能的开启和关闭。任何实现了 MessagePort 接口的特性，都可以独立完成消息的接收和发送，如共享线程中的 port 特性。接口清单如下：

```
interface MessagePort : EventTarget {
    void postMessage(any message, optional sequence<Transferable>
    transfer);
    void start();
    void close();
    //event handlers
    [TreatNonCallableAsNull] attribute Function? onmessage;
```

```
};
MessagePort implements Transferable;
```

由于 Worker 对象和 SharedWorker 对象中的 port 属性都属于 MessagePort 接口，因此这里介绍一下该接口中的方法和事件。

MessagePort 接口中包含了常用的 postMessage()方法和 message 事件，还包含了 start()方法和 close()方法。

- start()方法：开始调度从端口中接收的消息。如在介绍共享线程的时候，使用了 start()方法来启动添加的监听事件。
- end()方法：断开端口，使其不再活跃。这也是相对于共享线程而言的，即是断开了连接，而不是终止线程。

start()方法和 close()方法可用于共享线程，通过 port 属性调用，但在专属线程中不能使用。

16.4.2　线程内部的接口

线程内部的接口，就是线程的全局范围或线程中可进行的操作，主要描述在 Web 线程中（专属线程或共享线程），可以操作哪些对象、使用哪些功能。

由于 Web 线程是隐藏在背后执行的，在前面的介绍中，Web 线程是不可以访问窗体对象 window 和文档对象 document 的，因此，HTML 5 专门规范了 Web 线程的作用范围。

在前面的讲解中可以发现，专属线程和共享线程的线程脚本文件在写法上有着很大的不同，主要表现在线程文件的通信方面；除此之外，线程的作用范围有着很多共同之处。

线程内部的接口包含三个方面：通用线程范围接口、专属线程范围接口和共享线程范围接口。其中后面两个接口都实现了通用的接口。

1. WorkerGlobalScope通用线程范围接口清单

通用线程范围接口，是抽象的专属线程范围和共享线程范围的通用功能而形成的接口，定义了线程中一定要支持的功能。如是否在线、是否发生错误，以及 location 特性实现的功能。接口清单如下：

```
[NoInterfaceObject]
interface WorkerGlobalScope : EventTarget {
    readonly attribute WorkerGlobalScope self;
    readonly attribute WorkerLocation location;
    void close();
    [TreatNonCallableAsNull] attribute Function? onerror;
    [TreatNonCallableAsNull] attribute Function? onoffline;
    [TreatNonCallableAsNull] attribute Function? ononline;
};
WorkerGlobalScope implements WorkerUtils;
```

WorkerGlobalScope 接口定义了线程中所能使用的属性、方法和事件，并且实现了 WorkerUtils 接口。

self 属性：返回的是 WorkerGlobalScope 对象本身。在线程脚本文件中，通常是可以省略的。

location 属性：返回当线程被创建出来的时候与之关联的 WorkerLocation 对象，它表示用于初始化这个工作线程的脚本资源的绝对 URL，即使页面被多次重定向后，这个 URL 资源位置也不会改变。

close()方法：关闭线程。当脚本调用 WorkerGlobalScope 对象的 close()方法时，浏览器会自动执行两个步骤：首先，丢弃工作线程事件队列中的所有任务；其次，设置工作的 WorkerGlobalScope 对象的 closing 状态为 true（这将会阻止任何新的任务被加载到队列中来）。

error、online、offline 事件：分别是处理错误事件、在线事件和离线事件。

2. DedicatedWorkerGlobalScope专属线程范围接口清单

专属线程范围接口是在实现了通用线程范围接口的基础上，为专属线程范围提供的专门的接口。定义了专属线程中需要增加的发送消息和监听消息的功能。接口清单如下：

```
interface DedicatedWorkerGlobalScope {
    void postMessage(any message, optional sequence<Transferable>
    transfer);
    [TreatNonCallableAsNull] attribute Function? onmessage;
};
DedicatedWorkerGlobalScope implements WorkerGlobalScope;
```

DedicatedWorkerGlobalScope 接口实现了 WorkerGlobalScope 接口，另外隐含地实现了 MessagePort 对象，即增加了 postMessage()方法和 message 事件，用于在专属线程中发送消息和监听消息。

3. SharedWorkerGlobalScope共享线程范围接口清单

共享线程范围接口是在实现了通用线程范围接口的基础上，为共享线程范围提供的专门的接口。定义了共享线程中需要增加的连接事件、应用缓存和线程名称等功能。接口清单如下：

```
interface SharedWorkerGlobalScope : WorkerGlobalScope {
    readonly attribute DOMString name;
    readonly attribute ApplicationCache applicationCache;
    [TreatNonCallableAsNull] attribute Function? onconnect;
};
SharedWorkerGlobalScope implements WorkerGlobalScope;
```

SharedWorkerGlobalScope 接口实现了 WorkerGlobalScope 接口。共享线程的消息传递是在各自的连接中进行的，每个连接可以在 connect 事件中获取。

- ❑ name 属性：是创建 SharedWorkerGlobalScope 对象时，被指派的名称。该名称通常是指构造函数中指派的。
- ❑ applicationCache 属性：是一个应用缓存的网络模型，返回的是 ApplicationCache 对象。共享线程是它的缓存宿主。
- ❑ connect 事件：当共享线程有新的连接时触发。

4. WorkerUtils接口清单

WorkerUtils 接口是通用线程范围实现的接口，定义了专属线程范围和共享线程范围都

必须实现的功能。如 importScripts()方法、时间控制等功能。接口清单如下：

```
[NoInterfaceObject]
interface WorkerUtils {
    void importScripts(DOMString... urls);
    readonly attribute WorkerNavigator navigator;
};
WorkerUtils implements WindowTimers;
WorkerUtils implements WindowBase64;
```

其中，

- importScripts()方法：用于导入其他线程的脚本文件。
- navigator 属性：返回的是 WorkerNavigator 对象。

文档接口（如 Node 对象和 Document 对象等），是不能在线程中操作的。WorkerUtils 接口实现了 WindowTimers 和 WindowBase64。其中 WindowTimers 接口中定义了 setTimeout()，clearTimeout()，setInterval()，clearInterval()等方法。

5. WorkerLocation接口清单

WorkerLocation 接口是通用线程范围接口中的 location 特性所属的接口，定义了工作线程脚本资源的消息信息。接口清单如下：

```
interface WorkerLocation {
    //URL decomposition IDL attributes
    stringifier readonly attribute DOMString href;
    readonly attribute DOMString protocol;
    readonly attribute DOMString host;
    readonly attribute DOMString hostname;
    readonly attribute DOMString port;
    readonly attribute DOMString pathname;
    readonly attribute DOMString search;
    readonly attribute DOMString hash;
};
```

WorkerLocation 对象表示了工作线程脚本资源的绝对 URL 信息。我们可以使用它的 href 属性取得这个对象的绝对 URL。WorkerLocation 接口还定义了与位置信息有关的其他属性，如用于信息传输的协议（protocol）、主机名称（hostname）、端口（port）、路径名称（pathname）等。

16.5　小　结

本章主要讲解了 HTML 5 Web Workers 线程的概念，重点讲解了专属线程的基本用法、线程的嵌套、共享线程的基本用法和 Web 线程接口框架解析。本章的难点在于对 Web 线程概念的理解、专属线程和共享线程的区别，以及线程的嵌套。线程的作用范围也是新的内容，调试比较麻烦，也算是本章的难点。为了更加清晰地展现 Web 线程的完整框架，本章专门对 Web 线程框架接口做了详细的解析。对于前端开发人员来说，本章的内容确实是一个挑战。

下一章将介绍 HTML 5 的地理定位——Geolocation API。

16.6　习　　题

【习题 1】Web Workers 含了哪两种线程？

【习题 2】使用 Web Workers 多线程，需要提供相应的线程文件。在线程文件中，下列哪些功能不能使用（多选）：

A．document.getElementById()　　B．setTimeout()

C．window.history.back()　　D．setInterval ()

E．importScripts()　　F．alert()

【习题 3】Worker 对象可以创建专属线程，举例说明如何接收来自线程的消息。

【习题 4】SharedWorker 对象可以创建共享线程，举例说明如何向共享线程发送消息。

第 17 章　我知道你在哪里——地理位置 API

假设一个这样的场景，如果你的朋友向 Web 应用程序提供了自己的地理位置，那么你就可以在地图上很容易地找到他。在传统的 Web 应用中，要想获取用户的地理位置是十分困难的。随着越来越多的人拥有智能手持设备，地理位置逐渐成为最常用的应用之一。HTML 5 提供的 Geolocation API 让我们可以直接使用 JavaScript 脚本来获取地理位置信息。本章将详细介绍地理位置的相关概念及 Geolocation API 的用法。

17.1　Geolocation 概述

Geolocation API 定义了一个基于主机设备实现的高层次接口，用于获取地理位置信息，如经度和纬度等，API 本身无法知道地理位置信息的来源。如果使用基于地理位置的应用，则需要经过用户的授权。

17.1.1　地理位置信息

当使用 Geolocation API 请求用户的地理位置信息时，如果用户同意，浏览器就会返回相应的地理位置信息，该地理位置信息来源于用户设备提供的功能。

1. 地理位置坐标信息

这里介绍的地理位置坐标来自于世界大地测量系统（World Geodetic System），仅用于地球上的定位。最基本的地理位置坐标包括：

- 纬度
- 经度

例如，杭州的纬度为 30.273978，经度为 120.155361。在地理位置的应用中通常是以十进制格式来表示经纬度的。除了纬度和经度，完整的地理位置坐标信息还包括：

- 纬度和经度的精确度
- 海拔高度
- 海拔精确度
- 移动方向
- 移动速度

根据终端设备的功能不同，地理位置信息的完整性会有所差异。如果这些数据不存在，则返回 null。

2. 地理位置信息的来源

HTML 5 Geolocation API 并没有指定地理位置信息的来源，以及使用何种方式获取地理位置信息，这部分工作是由浏览器完成的。浏览器收到地理位置信息请求时，会访问地理位置信息的来源，以获取相应的地理位置信息。常见的地理位置信息来源如下。

- GPS（全球定位系统）。
- 网络信号位置，如 IP 地址、RFID、Wi-Fi、蓝牙的 MAC 地址。
- GSM/CDMA 手机的 ID。
- 用户自定义数据。

至于使用哪种地理位置信息来源，则取决于用户终端设备的功能。有些移动设备会支持 GPS、Wi-Fi、蓝牙等功能，但台式机则一般仅支持 IP 地址。所以，在实际的应用中，并不能保证用户设备返回的实际位置是精确的。

17.1.2　使用案例

基于地理位置的应用非常广泛，下面提供了一些使用案例。

1. 发现周围有趣的地方

当你去一个陌生的城市旅游时，你可以使用地理信息 API，Web 应用程序会根据你的大致位置，来呈现比较符合你的需要的结果。例如，你想查找或浏览周边的旅游景点、饭店、宾馆等信息，会变得十分方便。

2. 标注内容和位置信息

经常去旅行的人，通常会喜欢在走过的地方留下自己的足迹，如记录一些简短的文字或者图片并存储。每当增加新的内容时，Web 应用程序会自动地把地理位置一并保存。如果把记录的内容自动上传到博客等网站，就可以把自己的足迹分享给网络中的其他用户。

3. 确定自己的位置

当用户身处在陌生的城市或区域时，可能需要检测自己所处的位置。当他使用手持设备导航到一个基于 Web 地图的应用程序时，可以使用地理位置 API 来确定自己在地图上的确切位置。如果可能，Web 应用程序还能为他提供到达目的地的路线。

4. 路线导航

用户可以使用实时的地理位置信息，来跟踪地理位置的变化。可以使用地理位置 API 来重复更新变化的位置信息。

5. 监控有趣的地方

一个导游的 Web 应用，可以使用地理位置 API，来发现用户附近的有趣的地方，并触发音频或视频通知。当用户接近某个任务相关的地标时，就会触发提醒。

6. 及时更新本地信息

如果用户到达一个地方，需要获取本地的天气预报或新闻之类的信息，则可以使用地理定位 API。如果用户的位置发生变化，则相应的天气预报或新闻之类也随之发生变化。

7. 基于地理位置的社交网络

用户可以在 Web 应用中使用地理定位 API，及时地共享自己的位置信息，也可以跟踪自己的朋友网络。如果想约朋友一起喝茶，可以找一个比较方便的茶馆。

17.1.3　隐私策略

地理位置信息的泄漏会损害用户的隐私。Geolocation API 明确规范了用户隐私的保护机制：即通过 Geolocation API 获取用户位置信息，如果用户没有明确许可，是不能获取位置信息的。

1. 地理定位API中的隐私保护

在 Geolocation API 的规范中，地理位置的隐私保护机制需遵循如下策略。

- ❑ 在没有用户许可的情况下，浏览器不能向 Web 站点发送地理位置信息。
- ❑ 浏览器必须通过一个用户界面向用户提示以获得许可；或者该站点已经与用户建立了信任关系。
- ❑ 用户界面必须包括 URI 的主机部分。
- ❑ 如果页面跳转到其他没有许可的页面，则需要撤销保存的会话和许可。

对于一些用户代理将预先建立信任关系，不再需要上述的用户界面。例如，浏览到一个网页，如果网站执行地理位置请求，VOIP 电话可能不存在任何用户界面，就产生了地理位置许可。



图 17-1　在 Chrome 浏览器中询问是否共享地理位置

2. 隐私事项

应用程序只能在必要的时候请求地理位置信息；应用程序只能在提供给它们的任务中使用地理位置信息；一旦任务完成，应用程序必须释放地理位置信息，除非用户明确许可，否则不能保存位置信息；应用程序还必须防止未经授权的访问；如果地理位置信息被保存，则应允许用户删除或更新该信息。

应用程序不能转发未经用户许可的地理位置信息，提倡重传时使用加密的方式。在收集地理位置数据时，应用程序应该明确、清楚地披露以下内容。

- ❑ 会收集位置数据。

- 为什么收集位置数据。
- 数据将会被保留多久。
- 如何保证数据安全。
- 位置数据如何共享。
- 用户如何访问、更新和删除以及用户方面的任何其他选择。

3．一些特殊情况

在一些情况下，用户可能会无意中向浏览器授予许可，并透露自己的位置。或者是用户授权的是一个特定的 URL，由于内容发生变化，先前的授权应该不再适用；或者是用户可能仅仅是改变了主意。预测和防止这些情况是非常困难的，Geolocation 规范中也没有规定。然而应尽可能地方便用户撤销许可权限。

17.2　Geolocation 基本用法

本节将详细介绍 Geolocation 的使用方法，主要包括两种类型的地理位置请求：单次地理位置请求和重复性的地理位置更新。

17.2.1　浏览器的支持情况

Geolocation 是 HTML 5 中新增的一项重要功能。在各个浏览器的后来的版本中，开始陆续地支持地理位置的应用。所以，在使用基于地理位置的应用时，需检查浏览器的支持情况。

1．浏览器支持情况

各种浏览器对 HTML 5 Geolocation 的支持程度不同，并且还在不断更新。在 HTML 5 的所有新增功能中，Geolocation 是第一批被全部接受和实现的功能之一，相关的规范也已经达到相对成熟的阶段。对于开发人员来说，不用太担心未来的变化。目前 W3C 地理位置 API 被以下桌面浏览器支持：

- Firefox 3.5+
- Chrome 5.0+
- Safari 5.0+
- Opera 10.6+
- Internet Explorer 9.0+

W3C 地理位置 API 还可以被手机设备支持，如下：

- Android 2.0+
- iPhone 3.0+
- Opera Mobile 10.1+
- Symbian (S60 3rd & 5th generation)
- Blackberry OS 6

- ❑ Maemo

2. 浏览器支持性检查

由于浏览器的支持程度不同，因此在使用 Geolocaton 之前，有必要先检查浏览器是否支持该功能。通常是直接检测 navigator.geolocation 对象是否存在，这是一种最为便捷的检测方法。

【示例 17-1】 浏览器支持性检测。

```
if (navigator.geolocation) {
    alert("浏览器支持 Geolocation API");
} else {
    alert("浏览器不支持 Geolocation API");
}
```

检测浏览器是否支持 Geolocation API，可以减少不必要的错误。

17.2.2　单次获取地理位置

在许多的应用中，只请求一次用户的地理位置即可。单次请求地理位置，使用的是地理位置 API（即 navigator.geolocation 对象）的 getCurrentPosition()方法。

1. 使用getCurrentPosition()方法请求地理位置

使用 getCurrentPosition()方法，可以直接请求地理位置。该方法至少要有一个参数，这个必需的参数是一个回调处理函数。当地理位置获取成功时，会把地理位置信息交由回调函数来处理。

完整的使用方法如下所示。

【示例 17-2】 单次地理位置请求。

```
//成功回调函数：position 中包含了所有的地理位置信息
function successCallback(position){
    //显示 position 中的地理位置信息
}
//错误回调函数：error 中包含了错误的信息
function errorCallback(error){
    //显示 error 中的错误信息
}
var options = {};   //可选参数
if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition(successCallback,errorCallba
ck,options);        //单次请求
}
```

说明： 对象 navigator.geolocation 通过 getCurrentPosition()方法向浏览器底层设备请求地理位置信息，该方法有三个参数 successCallback、errorCallback 和 options，其中第一个参数是必需的，第二个和第三个参数是可选的。

如果请求成功，则调用回调函数 successCallback()；如果请求失败，则调用回调函数

errorCallback()；参数 options 则是在请求的过程中附件一些特性。

一旦调用了 getCurrentPosition()方法，浏览器就会询问用户是否允许透露地理位置信息，如图 17-1 所示。

2．回调函数successCallback()

在 getCurrentPosition()方法中，successCallback()函数是必需的回调函数。一旦地理位置信息请求成功，就会调用 successCallback()函数，该函数中的参数 position 包含了请求到的地理位置信息。参数 position 包含了一个位置坐标的特性 coords，该特性包含 7 个方面的信息：

- latitude（纬度）
- longitude（经度）
- accuracy（纬度和经度的精确度）
- altitude（海拔高度）
- altitudeAccuracy（海拔精确度）
- heading（移动方向）
- speed（移动速度）

其中前三个方面（latitude、longitude 和 accuracy）的信息是必需的数据，其他信息不能保证浏览器都能支持，如果不支持则返回 null。接下来完善一下回调函数 successCallback()。

【示例 17-3】 回调函数 successCallback()。

```
function successCallback(position){
    //显示 position 中的信息
    var temp = "<table border='0' cellpadding='0' cellspacing='1'>";
    temp += "<tr><td>纬度</td><td>" + position.coords.latitude +
    "</td></tr>";
    temp += "<tr><td>经度</td><td>" + position.coords.longitude +
    "</td></tr>";
    temp += "<tr><td>经纬精度</td><td>" + position.coords.accuracy +
    "</td></tr>";
    temp += "<tr><td>海拔</td><td>" + position.coords.altitude +
    "</td></tr>";
    temp += "<tr><td>海拔精度</td><td>" + position.coords.altitudeAccuracy
    + "</td></tr>";
    temp += "<tr><td>前进方向</td><td>" + position.coords.heading +
    "</td></tr>";
    temp += "<tr><td>移动速度</td><td>" + position.coords.speed +
    "</td></tr>";
    temp += "</table>";
    document.getElementById("resultText").innerHTML = temp;
}
```

将会得到如图 17-2 所示的地理位置信息，获取到了最基本的纬度、经度和纬度和经度的精确度。

3．回调处理函数errorCallback()

在 getCurrentPosition()方法中，errorCallback ()函数是可选的回调函数。在地理位置信

息请求过程中，如果包含了 errorCallback ()函数，则只要请求不成功，都会调用回调函数 errorCallback()。这其中也包括未被用户许可。

errorCallback()回调函数包含一个 error 参数，该参数中记录了错误的代码（code 特性）和错误信息（message 特性）。其中错误代码中的错误编号代表了不同的意义，错误编号如下。

- ❑ PERMISSION_DENIED（编号为 1）：表示用户选择拒绝浏览器获得其地理位置信息。
- ❑ POSITION_UNAVAILABLE（编号为 2）：表示已尝试获取用户的地理位置，但失败了。
- ❑ TIMEOUT（编号为 3）：如果用户设置了可选的 timeout 值，则当尝试请求用户的地理位置超过该时间时，意味着超时。

HTML5 Geolocation

纬度	30.273978
经度	120.155361
经纬精度	25000
海拔	null
海拔精度	null
前进方向	null
移动速度	null

图 17-2　获取到的地理位置信息

在出现错误的情况下，需要让用户知道应用程序出了问题。而当获取失败或请求超时的时候，通常会希望再一次尝试请求。接下来完善一下回调函数 errorCallback ()。

【示例 17-4】 回调函数 errorCallback()。

```
function errorCallback(error){
    //显示 error 中的错误信息
    var err = "";
    switch(error.code){
        case error.PERMISSION_DENIED:
            err = "用户阻止了该页面获取地理位置:" + error.message;
            alert(err);
            break;
        case error.POSITION_UNAVAILABLE:
            err = "浏览器没能获取到地理位置：" + error.message + "\n 是否尝试再
            次请求？";
            confirm(err)?navigator.geolocation.getCurrentPosition
            (successCallback,errorCallback):"";
            break;
        case error.TIMEOUT:
            err = "获取地理位置超时：" + error.message + "\n 是否尝试再次请求？";
            confirm(err)?navigator.geolocation.getCurrentPosition
             (successCallback,errorCallback):"";
            break;
        default:
            err = "获取地理位置时，产生了一个错误：" + error.message;
            alert(err);
            break;
    }
}
```

4．可选参数options

在 getCurrentPosition()方法中，options 参数是可选的。options 是一个对象参数，其中包括了三个可选的特性，分别介绍如下。

- enableHighAccuracy：参数默认为 false。如果设置为 true，则会通知浏览器启用 Geolocation 的高精确度模式。
- timeout：可选值，单位为 ms，默认为 Infinity（即无穷大的意思）。设置当前位置请求所允许的最长时间，如果在这个时间段内没有完成，则会调用错误处理的回调函数。询问用户许可的时间也包括在内。
- maximumAge：可选值，单位为 ms，默认为 0。设置浏览器重新计算地理位置的时间间隔，即更新位置信息的频率。只要浏览器在该时间段之内成功请求过位置信息，就不会重新计算位置，直接使用之前获取的位置信息。

接下来完善一下可选参数 options。

【示例 17-5】 可选参数 options。

```
var options = {
    timeout : 5000,
    maximumAge : 600000
}; //可选参数
```

5. Geolocation单次请求示例

综合前面介绍的内容，就可以构建一个完整的获取地理位置的应用。下面是完整的示例代码。

【示例 17-6】 可选参数 options。

```
<!-- 页面信息 -->
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<title>HTML 5 Geolocation</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div class="content">
  <h1> HTML 5 Geolocation </h1>
  <p id="message"></p>
  <div id="resultText">
  </div>
  <div style="clear:both;"></div>
</div>
</body>
</html>
<!-- 脚本信息 -->
<script type="text/javascript">
//回调函数：position 中包含了所有的地理位置信息
function successCallback(position){
    //显示 position 中的信息
    var temp = "<table border='0' cellpadding='0' cellspacing='1'>";
    temp += "<tr><td>纬度</td><td>" + position.coords.latitude +
    "</td></tr>";
    temp += "<tr><td>经度</td><td>" + position.coords.longitude +
    "</td></tr>";
    temp += "<tr><td>经纬精度</td><td>" + position.coords.accuracy +
    "</td></tr>";
    temp += "<tr><td>海拔</td><td>" + position.coords.altitude +
```

```
    "</td></tr>";
    temp += "<tr><td>海拔精度</td><td>" + position.coords.altitudeAccuracy
    + "</td></tr>";
    temp += "<tr><td>前进方向</td><td>" + position.coords.heading +
    "</td></tr>";
    temp += "<tr><td>移动速度</td><td>" + position.coords.speed +
    "</td></tr>";
    temp += "</table>";
    document.getElementById("resultText").innerHTML = temp;
}
//错误回调函数：error 中包含了错误的信息
function errorCallback(error){
    //显示 error 中的错误信息
    var err = "";
    switch(error.code){
        case error.PERMISSION_DENIED:
            err = "用户阻止了该页面获取地理位置:" + error.message;
            alert(err);
            break;
        case error.POSITION_UNAVAILABLE:
            err = "浏览器没能获取到地理位置：" + error.message + "\n 是否尝试再
            次请求？";
            confirm(err)?navigator.geolocation.getCurrentPosition
            (successCallback,errorCallback,options):"";
            break;
        case error.TIMEOUT:
            err = "获取地理位置超时：" + error.message + "\n 是否尝试再次请求？";
            confirm(err)?navigator.geolocation.getCurrentPosition
            (successCallback,errorCallback,options):"";
            break;
        default:
            err = "获取地理位置时，产生了一个错误：" + error.message;
            alert(err);
            break;
    }
}
var options = {
    timeout : 5000,
    maximumAge : 600000
}; //可选参数
if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition(successCallback,errorCallba
ck,options);   //单次请求
}
</script>
```

成功获取地理位置的运行结果如图 17-2 所示。

17.2.3　重复性的位置信息更新

但是在一些应用中，需要不断地更新用户的地理位置信息，HTML 5 Geolocation 为我们提供了重复更新地理位置信息的方法 watchPosition()。watchPosition()和 getCurrent-Position()的使用方法完全类似：

```
navigator.geolocation.getCurrentPosition(successCallback,errorCallback,
options);
```

```
var watchId = navigator.geolocation.watchPosition(successCallback,
errorCallback,options);
```

仅有一点不同，使用 watchPosition()方法会首先返回一个 watchId，然后以既定时间间隔不断地请求位置信息。如果用户想关闭更新，可以使用 clearWatch()方法，用法如下：

```
navigator.geolocation.clearWatch(watchId);
```

需要说明的是，在 watchPosition()方法中设置的 timeout 特性，是指每一次请求位置信息时的过期时间，而不是总的过期时间；位置信息更新的频率会参考 maximumAge 特性；当再次请求位置信息时，只有当请求的位置发生变化的时候，才会调用回调函数 successCallback()。

17.3　Geolocation 接口解析

在 Geolocation API 中，涉及多个接口对象。为了能理清它们之间的关系，本节将详细讲解地理位置 Geolocation 接口及其附属接口。

1. NavigatorGeolocation接口清单

NavigatorGeolocation 接口定义的是一个由浏览器的 navigator 对象实现的接口。接口清单如下：

```
[NoInterfaceObject]
interface NavigatorGeolocation {
    readonly attribute Geolocation geolocation;
};
Navigator implements NavigatorGeolocation;
```

NavigatorGeolocation 接口中有一个属于 Geolocation 对象的 geolocation 特性。由于 Navigator 实现了这个接口，因此我们可以使用 navigator.geolocation，来访问 Geolocation 对象及其相关的附属对象。

2. Geolocation接口清单

Geolocation 接口是地理位置应用的核心接口，定义了获取地理位置的 3 个重要的方法和 2 个回调处理函数。接口清单如下：

```
[NoInterfaceObject]
interface Geolocation {
    void getCurrentPosition(in PositionCallback successCallback,
                            in optional PositionErrorCallback errorCallback,
                            in optional PositionOptions options);
    long watchPosition(in PositionCallback successCallback,
                            in optional PositionErrorCallback errorCallback,
                            in optional PositionOptions options);
    void clearWatch(in long watchId);
};
[Callback=FunctionOnly, NoInterfaceObject]
interface PositionCallback {
    void handleEvent(in Position position);
};
```

```
[Callback=FunctionOnly, NoInterfaceObject]
interface PositionErrorCallback {
    void handleEvent(in PositionError error);
};
```

由接口清单可知，Geolocation 接口提供了 3 个方法：分别为 getCurrentPosition()、watchPosition()和 clearWatch()。

而对于前两个方法中使用的回调函数 successCallback()有一个 Position 对象的参数 position；回调函数 errorCallback()有一个 PositionError 对象的参数 error。可选参数 options 则属于 PositionOptions 对象。

3. PositionOptions接口清单

PositionOptions 接口定义了获取地理位置时的可选参数。在 Geolocation 接口中，获取地理位置时，包含一个可选参数 options，该参数从属于 PositionOptions 接口。PositionOptions 接口清单如下：

```
[Callback, NoInterfaceObject]
interface PositionOptions {
    attribute boolean enableHighAccuracy;
    attribute long timeout;
    attribute long maximumAge;
};
```

PositionOptions 接口中有 3 个特性，在前面介绍可选参数 options 时已经介绍过了。

4. Position接口清单

Position 接口定义了地理位置信息的结构。主要包含一个坐标和一个获取该坐标的时间戳。接口清单如下：

```
[NoInterfaceObject]
interface Position {
    readonly attribute Coordinates coords;
    readonly attribute DOMTimeStamp timestamp;
};
```

Position 接口的两个特性如下。

- ❑ coords：从属于 Coordinates 对象，表示位置信息中的坐标信息。在坐标信息中，不仅包含了纬度、经度、准确度，还包含了海拔、海拔准确度、移动方向和移动速度等。
- ❑ timestamp：时间戳，即采集到位置信息的时间点。

5. Coordinates接口清单

Coordinates 接口定义了地理位置坐标的详细信息，全方位描述了地理位置的各个特征。接口清单如下：

```
[NoInterfaceObject]
interface Coordinates {
    readonly attribute double latitude;
    readonly attribute double longitude;
    readonly attribute double? altitude;
```

```
    readonly attribute double accuracy;
    readonly attribute double? altitudeAccuracy;
    readonly attribute double? heading;
    readonly attribute double? speed;
};
```

Coordinates 接口的 7 个特性分别描述了地理位置的 7 个方面的信息：纬度（latitude）、经度（longitude）、经纬度的精确度（accuracy）、海拔（altitude）、海拔精确度（altitudeAccuracy）、移动方向（heading）和移动速度（speed）。

6．PositionError接口清单

PositionError 接口，定义了一个错误的信息结构。当获取地理位置发生错误时，错误处理函数会接收这样的错误信息，包含了错误编码和详细的错误描述。接口清单如下：

```
[NoInterfaceObject]
nterface PositionError {
    const unsigned short PERMISSION DENIED = 1;
    const unsigned short POSITION UNAVAILABLE = 2;
    const unsigned short TIMEOUT = 3;
    readonly attribute unsigned short code;
    readonly attribute DOMString message;
};
```

PositionError 接口中有两个特性和 3 个常量。

- code：表示错误代码。其中有 3 个错误代码被定义为常量：PERMISSION_DENIED 表示用户拒绝了地理位置请求；POSITION_UNAVAILABLE 表示获取地理位置时失败；TIMEOUT 表示获取地理位置超时。
- message：描述了发生错误的详细信息。此特性主要用于开发人员的调试，不应该把这些错误消息显示到用户界面。

17.4　实验室：在地图上显示我的位置

本节将使用 Geolocation API 实现一个常用的应用，就是把获取到的地理位置显示的地图上。而地图应用则来源于地图服务商，目前网络上可以使用的地图服务有很多，这里选择谷歌地图服务。

1．案例简介

在本节介绍的案例中，首先，通过 Geolocation API 获取用户当前所在位置；然后，把地理位置数据传递给调用的地图服务；最终，在地图上标注用户当前所处的位置，案例效果如图 17-3 所示。

2．设计页面

首先设计一个简易的页面，主要包括标题、黄色背景的提示信息、地理位置数据、标注了“我的位置”的地图。

【示例 17-7】　在地图上标注“我的位置”。



图 17-3　在地图上标注我的位置

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<title>HTML 5 Geolocation</title>
<link href="style.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=false"></script>
</head>
<body>
<div class="content">
  <h1> HTML 5 Geolocation </h1>
  <p class="message" id="message">HTML 5 Geolocation</p>
  <div id="resultText"> 位置数据 </div>
  <div id="resultMap"> 地图 </div>
  <div style="clear:both;"></div>
</div>
</body>
</html>
```

示例 17-7 代码清单中，引用了 CSS 样式表（不再介绍）和一个链接到谷歌地图服务的脚本文件：

```
<script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=false"></script>
```

src 特性引用的地址，可以链接到谷歌的地图服务，我们就在这个基础上完成地图应用的功能。

3．编写基本的位置请求脚本

这里使用的是单次地理位置请求。

首先定义一个全局变量 goe，再定义一个获取 Geolocation 对象的函数 getGeolocation()。

我们在页面加载完成的时候执行地理位置请求。

```
<script type="text/javascript">
var geo; //全局变量
//获取 Geolocation 对象
function getGeolocation(){
    try{
        if(!!navigator.geolocation) return navigator.geolocation;
        else return undefined;
    }catch(e){
        return undefined;
    }
}
window.onload = function(){
    if((geo=getGeolocation())){
        document.getElementById("message").textContent = "正在使用 HTML 5 
        地理定位...";
        geo.getCurrentPosition(geo_success,geo_error,options);
    } else {
        document.getElementById("message").textContent = "不支持 HTML 5 地
        理定位！";
    }
}
</script>
```

4. 设置可选参数options

设置 5 秒钟的超时限制，以及 10 分钟的位置更新频率。

```
<script type="text/javascript">
var options = {
    enableHighAccuracy : false,
    timeout : 5000,          //超时时间限制
    maximumAge : 600000      //位置更新频率
};
</script>
```

5. 编写成功获取地理位置的回调函数geo_success()

如果成功获取到用户的地理位置，则一方面显示用户得到的位置数据，另一方面去调用 Google 地图 API，并在地图上标注刚刚获取到的位置信息。

```
<script type="text/javascript">
function geo_success(position){
    //开始在页面上显示地理位置数据
    var temp = "<table border='0' cellpadding='0' cellspacing='1'>";
    temp += "<tr><td>纬度</td><td>" + position.coords.latitude + 
    "</td></tr>";
    temp += "<tr><td>经度</td><td>" + position.coords.longitude + 
    "</td></tr>";
    temp += "<tr><td>经纬精度</td><td>" + position.coords.accuracy + 
    "</td></tr>";
    temp += "<tr><td>海拔</td><td>" + position.coords.altitude + 
    "</td></tr>";
    temp += "<tr><td>海拔精度</td><td>" + position.coords.altitudeAccuracy 
    + "</td></tr>";
```

```
    temp += "<tr><td>前进方向</td><td>" + position.coords.heading +
    "</td></tr>";
    temp += "<tr><td>移动速度</td><td>" + position.coords.speed +
    "</td></tr>";
    temp += "</table>";
    document.getElementById("resultText").innerHTML = temp;

    //开始调用 Google 地图 API
    var latlng = new google.maps.LatLng(position.coords.latitude,
    position.coords.longitude);
    var myOptions = {
        zoom: 15, center: latlng,
        mapTypeControl: false,
        navigationControlOptions: {
            style: google.maps.NavigationControlStyle.SMALL
        },
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("resultMap"),
    myOptions);
    var marker = new google.maps.Marker({ position: latlng, map: map, title:"
    我的位置!" });
}
</script>
```

关于 Google 地图 API，可以去 Google 网站参考相关的文档说明。

6．编写错误回调函数geo_error()

当出现错误或用户拒绝时，会把相应的错误提示信息显示到页面中的黄色提示区域。如果发生的错误是 POSITION_UNAVAILABLE 或 TIMEOUT，将会询问是否重新获取地理位置。

```
<script type="text/javascript">
function geo_error(error){
    switch(error.code){
        case error.PERMISSION_DENIED:
            document.getElementById("message").textContent = "用户阻止了获
            取地理位置。";
            break;
        case error.POSITION_UNAVAILABLE:
            document.getElementById("message").textContent = "浏览器没能获
            取到地理位置。";
            confirm("是否尝试再次请求？")?geo.getCurrentPosition(geo
            success,geo error,options):"";
            break;
        case error.TIMEOUT:
            document.getElementById("message").textContent = "获取地理位置
            超时。";
            confirm("是否尝试再次请求？")?geo.getCurrentPosition(geo_
            success,geo_error,options):"";
            break;
        default:
            document.getElementById("message").textContent = "产生了一个未
            知的错误。";
            break;
    }
}
```

```
</script>
```

这样，就可以在地图上标注“我的位置”了，示例运行结果如图 17-3 所示。

7．改进的应用

我们可以把它改进为不断更新地理位置应用，使用 watchPosition()方法定时重复更新移动变化的地理位置。首先定义一个全局变量 watchId，并把 window.onload 中的代码改进如下：

```
var watchId;
window.onload = function(){
    if((geo=getGeolocation())){
        document.getElementById("message").textContent = "正在使用 HTML 5
        地理定位...";
        watchId = geo.watchPosition (geo_success,geo_error,options);
    } else {
        document.getElementById("message").textContent = "不支持 HTML 5 地
        理定位！";
    }
}
```

这样，如果“我的位置”处在变化之中，地图就会按照一定的频率不断获取新的位置，并更新地图中的标注。

如果需要关闭频繁的地理位置获取，可以调用下面的代码：

```
geo.clearWatch(watchId);
```

改进后的应用对于台式机等固定设备没有任何意义。如果是使用有 GPS 功能的移动设备，则可以实时地监控用户当前的位置。

17.5　小　　结

本章主要讲解了 HTML 5 Geolocation 获取地理位置的功能。重点讲解了地理位置的坐标信息、地理位置获取原理、隐私策略和 Geolocation 的用法，并详细解析了 Geolocation 接口。本章的难点在于对各个接口对象的理解，由于对象直接相互嵌套，因此在使用过程中很容易出现写法上的错误。本章的另一个难点是地理位置的获取比较难理解，主要是这一部分对我们是透明的，完全交给了浏览器来实现。

17.6　习　　题

【习题 1】请列举常用的地理位置信息来源，至少列出三种。

【习题 2】能否在不知不觉中获取用户地理位置？

【习题 3】在 Geolocation 的使用方法中，包含哪两种地理位置请求？

【习题 4】如果地理位置请求成功，则会返回一个位置坐标。请问该位置坐标包含哪些信息？

【习题 5】编写一个获取设备地理位置的功能，并结合地图服务显示获取的地理位置。